# TCP/IP Protocol Suite

## Fourth Edition

Behrouz A. Forouzan

# *Acronyms*

| | |
|---|---|
| **2DES** | double DES |
| **3DES** | triple DES |
| **AAL** | application adaptation layer |
| **ADSL** | asymmetric digital subscriber line |
| **AES** | Advanced Encryption Standard |
| **AH** | Authentication Header |
| **ANSI** | American National Standards Institute |
| **ANSNET** | Advanced Networks and Services Network |
| **ARP** | Address Resolution Protocol |
| **ARPA** | Advanced Research Projects Agency |
| **ARPANET** | Advanced Research Projects Agency Network |
| **AS** | autonomous system |
| **ASCII** | American Standard Code for Information Interchange |
| **ASN.1** | Abstract Syntax Notation 1 |
| **ATM** | asynchronous transfer mode |
| **BER** | Basic Encoding Rules |
| **BGP** | Border Gateway Protocol |
| **BOOTP** | Bootstrap Protocol |
| **BSS** | basic service set |
| **CA** | certification authority |
| **CBC** | cipher-block chaining |
| **CBT** | core-based tree |
| **CCITT** | Consultative Committee for International Telegraphy and Telephony |
| **CGI** | common gateway interface |
| **CIDR** | Classless Interdomain Routing |
| **CMS** | Cryptographic Message Syntax |
| **CSMA/CA** | carrier sense multiple access with collision avoidance |
| **CSMA/CD** | carrier sense multiple access with collision detection |
| **CSNET** | Computer Science Network |
| **DARPA** | Defense Advanced Research Projects Agency |
| **DDN** | Defense Data Network |
| **DDNS** | Dynamic Domain Name System |
| **DES** | Data Encryption Standard |
| **DHCP** | Dynamic Host Configuration Protocol |
| **DNS** | Domain Name System |
| **DSL** | digital subscriber line |
| **DSS** | Digital Signature Standard |
| **DSSS** | direct sequence spread spectrum |
| **DVMRP** | Distance Vector Multicast Routing Protocol |
| **EBCDIC** | Extended Binary Coded Decimal Interchange Code |
| **EIA** | Electronic Industries Alliance |

**ESP** Encapsulating Security Payload

**ESS** extended service set

**FCC** Federal Communications Commission

**FCS** frame check sequence

**FHSS** frequency hopping spread spectrum

**FQDN** fully qualified domain name

**FTP** File Transfer Protocol

**HDSL** high bit rate digital subscriber line

**HMAC** hashed message authentication code

**HTML** Hypertext Markup Language

**HTTP** Hypertext Transfer Protocol

**IAB** Internet Architecture Board

**IANA** Internet Assigned Numbers Authority

**ICANN** Internet Corporation for Assigned Names and Numbers

**ICMP** Internet Control Message Protocol

**IEEE** Institute of Electrical and Electronics Engineers

**IEGS** Internet Engineering Steering Group

**IETF** Internet Engineering Task Force

**IGMP** Internet Group Management Protocol

**IKE** Internet Key Exchange

**INTERNIC** Internet Network Information Center

**IP** Internet Protocol

**IPng** Internetworking Protocol, next generation

**IPSec** IP Security

**IRTF** Internet Research Task Force

**ISAKMP** Internet Security Association and Key Management Protocol

**ISO** International Organization for Standardization

**ISOC** Internet Society

**ISP** Internet service provider

**ITU-T** International Telecommunications Union–Telecommunication Standardization Sector

**IV** initial vector

**KDC** key-distribution center

**LAN** local area network

**LCP** Link Control Protocol

**LIS** logical IP subnet

**LSA** link state advertisement

**MAA** message access agent

**MAC** media access control or message authentication code

**MBONE** multicast backbone

**MD** Message Digest

**MIB** management information base

**MILNET** Military Network

**MIME** Multipurpose Internet Mail Extension

**MOSPF** Multicast Open Shortest Path First

**MSS** maximum segment size

**MTA** message transfer agent

**MTU** maximum transfer unit

**NAP** Network Access Point

**NAT** network address translation

**NIC** Network Information Center

**NIC** network interface card

**NIST** National Institute of Standards and Technology

**NSA** National Security Agency

| | |
|---|---|
| **NSF** | National Science Foundation |
| **NSFNET** | National Science Foundation Network |
| **NVT** | network virtual terminal |
| **OSI** | Open Systems Interconnection |
| **OSPF** | open shortest path first |
| **PGP** | Pretty Good Privacy |
| **PIM** | Protocol Independent Multicast |
| **PIM-DM** | Protocol Independent Multicast, Dense Mode |
| **PIM-SM** | Protocol Independent Multicast, Sparse Mode |
| **PING** | Packet Internet Groper |
| **PKI** | public-key infrastructure |
| **POP** | Post Office Protocol |
| **PPP** | Point-to-Point Protocol |
| **PQDN** | partially qualified domain name |
| **RACE** | Research in Advanced Communications for Europe |
| **RADSL** | rate adaptive asymmetrical digital subscriber line |
| **RARP** | Reverse Address Resolution Protocol |
| **RFC** | Request for Comment |
| **RIP** | Routing Information Protocol |
| **ROM** | read-only memory |
| **RPB** | reverse path broadcasting |
| **RPF** | reverse path forwarding |
| **RPM** | reverse path multicasting |
| **RSA** | Rivest, Shamir, Adelman |
| **RTCP** | Real-time Transport Control Protocol |
| **RTP** | Real-time Transport Protocol |
| **RTSP** | Real-Time Streaming Protocol |
| **RTT** | round-trip time |
| **S/MIME** | Secure/Multipurpose Internet Mail Extension |
| **SA** | security association |
| **SAD** | Security Association Database |
| **SCTP** | Stream Control Transmission Protocol |
| **SDH** | synchronous digital hierarchy |
| **SDSL** | symmetric digital subscriber line |
| **SFD** | start frame delimiter |
| **SHA** | Secure Hash Algorithm |
| **SI** | stream identifier |
| **SKEME** | Secure Key Exchange Mechanism |
| **SMI** | Structure of Management Information |
| **SNMP** | Simple Network Management Protocol |
| **SONET** | Synchronous Optical Network |
| **SP** | Security Policy |
| **SPD** | Security Policy Database |
| **SSH** | secure shell |
| **SSL** | Secure Sockets Layer |
| **SSN** | stream sequence number |
| **SVC** | switched virtual circuit |
| **TCP** | Transmission Control Protocol |
| **TCP/IP** | Transmission Control Protocol/ Internet Protocol |
| **TELNET** | Terminal Network |
| **TFTP** | Trivial File Transfer Protocol |

| | |
|---|---|
| **TLS** | Transport Layer Security |
| **TOS** | type of service |
| **TSN** | transmission sequence number |
| **TTL** | time to live |
| **UA** | user agent |
| **UDP** | User Datagram Protocol |
| **UNI** | user network interface |
| **URL** | uniform resource locator |
| **VC** | virtual circuit |
| **VCI** | virtual channel identifier |
| **VDSL** | very high bit rate digital subscriber line |
| **VPI** | virtual path identifier |
| **VPN** | virtual private network |
| **WAN** | wide area network |
| **WWW** | World Wide Web |

# TCP/IP

## Protocol Suite

## McGraw-Hill Forouzan Networking Series

Titles by Behrouz A. Forouzan:

*Data Communications and Networking*
*TCP/IP Protocol Suite*
*Local Area Networks*
*Business Data Communications*
*Cryptography and Network Security*

# TCP/IP

## Protocol Suite

Fourth Edition

Behrouz A. Forouzan

Higher Education

Boston Burr Ridge, IL Dubuque, IA New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto

The McGraw-Hill Companies

Higher Education

TCP/IP PROTOCOL SUITE, FOURTH EDITION

Published by McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 1221 Avenue of the Americas, New York, NY 10020. 

This book is printed on acid-free paper.

1 2 3 4 5 6 7 8 9 0 DOC/DOC 0 9

ISBN 978-0-07-337604-2
MHID 0-07-337604-3

Global Publisher: *Raghothaman Srinivasan*
Director of Development: *Kristine Tibbetts*
Senior Marketing Manager: *Curt Reynolds*
Project Manager: *Joyce Watters*
Senior Production Supervisor: *Sherry L. Kane*
Lead Media Project Manager: *Stacy A. Patch*
Designer: *Laurie B. Janssen*
Cover Designer: *Ron Bissell*
(USE) Cover Image: © *Chad Baker/Getty Images*
Compositor: *Macmillan Publishing Solutions*
Typeface: *10/12 Times Roman*
Printer: *R. R. Donnelley Crawfordsville, IN*



**Library of Congress Cataloging-in-Publication Data**

Forouzan, Behrouz A.
TCP/IP protocol suite / Behrouz A. Forouzan.—4th ed.
p. cm.
Includes index.
ISBN 978-0-07-337604-2—ISBN 0-07-337604-3 (hard copy : alk. paper) 1. TCP/IP (Computer network protocol) I. Title.
TK5105.585.F67 2010
004.6'2—dc22

2008051008

www.mhhe.com

*To the memory of my parents,*
*the source of my inspiration.*

—Behrouz A. Forouzan

# Brief Contents

# Contents

# Preface

Technologies related to networks and internetworking may be the fastest growing in our culture today. Many professors and students who have used, read, or reviewed the third edition of the book suggested the publication of a new edition that include these changes. In the fourth edition, I have reorganized the book incorporating many changes and added several new chapters and appendices.

The fourth edition of the book assumes the reader has no prior knowledge of the TCP/IP protocol suite, although a previous course in data communications is desirable.

## Organization

This book is divided into seven parts.

- Part I (Introduction and Underlying Technologies), comprising Chapters 1 to 3, reviews the basic concepts and underlying technologies that, although independent from the TCP/IP protocols, are needed to support them.
- Part II (Network Layer), comprising Chapters 4 to 12, discusses IPv4 addressing, the IPv4 protocol, all auxiliary protocols helping IPv4 protocol, and unicast and multicast routing protocols.
- Part III (Transport Layer), comprising Chapters 13 to 16, introduces the general concepts in the transport layer (Chapter 13) and then fully discusses three transport layer protocols: UDP, TCP, and SCTP (Chapters 14, 15, and 16).
- Part IV (Application Layer), comprising Chapters 17 to 25, introduces the general concepts in the application layer including client-server programming (Chapter 17) and then fully discusses seven application layer protocols (Chapters 18 to 24). Chapter 25 is devoted to multimedia in the Internet.
- Part V (New Generation), comprising Chapters 26 to 28, introduces the new generation of IP protocol, IPv6 addressing (Chapter 26), IPv6 protocol (Chapter 27), and ICMPv6 (Chapter 28).
- Part VI (Security), comprising Chapters 29 to 30, discusses the inevitable topics such as cryptography and network security (Chapter 29) and Internet security (Chapter 30).
- Part VII (Appendices) inclosed seven appendices that may be needed when reading the book.

## Features

Several features of this text are designed to make it particularly easy for students to understand TCP/IP.

### *Visual Approach*

The book presents highly technical subject matter without complex formulas by using a balance of text and figures. More than 650 figures accompanying the text provide a visual and intuitive opportunity for understanding the material. Figures are particularly important in explaining networking concepts, which are based on connections and transmission. Often, these are more easily grasped visually rather than verbally.

### *Highlighted Points*

I have repeated important concepts in boxes for quick reference and immediate attention.

### *Examples and Applications*

Whenever appropriate, I have included examples that illustrate the concepts introduced in the text. Also, I have added real-life applications throughout each chapter to motivate students.

### *Protocol Packages*

Although I have not tried to give the detailed code for implementing each protocol, many chapters contain a section that discusses the general idea behind the implementation of each protocol. These sections provide an understanding of the ideas and issues involved in each protocol, but may be considered optional material.

### *Key Terms*

The new terms used in each chapter are listed at the end of the chapter and their definitions are included in the glossary.

### *Summary*

Each chapter ends with a summary of the material covered by that chapter. The summary is a bulleted overview of all the key points in the chapter.

### *Practice Set*

Each chapter includes a practice set designed to reinforce salient concepts and encourage students to apply them. It consists of two parts: exercises and research activities. Exercises require understanding of the material. Research activities challenge those who want to delve more deeply into the material.

### *Appendices*

The appendices are intended to provide a quick reference or review of materials needed to understand the concepts discussed in the book. The appendices in the previous edition have been revised, combined, and some new ones have been added.

### *Glossary and Acronyms*

The book contains an extensive glossary and a list of acronyms.

### *Instructor Resources*

Solutions, PowerPoint presentations, and Student Quizzes are available through the book's website at www.mhhe.com/forouzan.

### *Electronic Book Options*

**CourseSmart.** This text is offered through CourseSmart for both instructors and students. CourseSmart is an online browser where students can purchase access to this and other McGraw-Hill textbooks in digital format. Through their browser, students can access a complete text online at almost half the cost of a traditional text. Purchasing the etextbook also allows students to take advantage of CourseSmart's Web tools for learning, which include full text search, notes and highlighting, and e-mail tools for sharing notes between classmates. To learn more about CourseSmart options, contact your sales representative or visit www.CourseSmart.com.

**VitalSource.** VitalSource is a downloadable eBook. Students who choose the VitalSource eBook can save up to 45 percent off the cost of the print book, reduce their impact on the environment, and access powerful digital learning tools. Students can share notes with others, customize the layout of the etextbook, and quickly search their entire etextbook library for key concepts. Students can also print sections of the book for maximum portability.

## New and Changes to the Fourth Edition

There are many changes and much new material in the fourth edition, including:

- Chapter objectives have been added to the beginning of each chapter.
- A brief references list and a list of corresponding RFCs have been added at the end of each chapter.
- Some new exercises and some research activities are added to some chapters.
- Figures are revised to reflect their relation to the actual technology used today.
- Chapter 3 (Underlying Technologies) has been totally revised to cover new technologies
- Chapter 4 (Introduction to Network Layer) is totally new.
- Chapter 13 (Introduction to the Transport Layer) is totally new.
- Chapter 17 (Introduction to the Application Layer) is totally new.
- Chapter 5 now discusses both classful and classless addressing (a combination of Chapters 4 and 5 in the third edition).
- Chapter 6 has been revised to include MPLS.
- Materials on New Generation Internet Protocol (IPv6) has been augmented to three chapters (Chapters 26, 27, 28).
- Materials on security have been augmented to two chapters (Chapters 29, 30).
- Some deprecated protocols, such as RARP and BOOTP are removed to provide space for new material.
- Chapters are reorganized according to the layers in TCP/IP protocol suite.
- Appendix A (ASCII Code) has been replaced by Unicode.
- Appendix C (Error Detection) has been totally revised and augmented.
- Appendix D (Checksum) is totally revised.

- Appendix E (HTML, XHTML, XML, and XSL) is totally new.
- Appendix F (Client-Server Programming in Java) is totally new.
- Appendix G (Miscellaneous Information) is now a combination of the previous Appendices F, G, and H.

## How to Use the Book

This book is written for both academic and professional audiences. The book can be used as a self-study guide for interested professionals. As a textbook, it can be used for a one-semester or one-quarter course. The chapters are organized to provide a great deal of flexibility. I suggest the following:

- Chapters 1 to 3 can be skipped if students have already taken a course in data communications and networking.
- Chapters 4 through 25 are essential for understanding the TCP/IP protocol suite.
- Chapters 26 to 28 can be used at the professor's discretion if there is a need for making the student familiar with the new generation.
- Chapters 29 and 30 can prepare the students for a security course, but they can be skipped if there is time restraint.

## Acknowledgments for the Fourth Edition

It is obvious that the development of a book of this scope needs the support of many people. I acknowledged the contributions of many people in the preface of the first three editions. For the fourth edition, I would like to acknowledge the contributions from peer reviewers to the development of the book. These reviewers are:

Dale Buchholz, *DePaul University*
Victor Clincy, *Kennesaw State University*
Richard Coppins, *Virginia Commonwealth University*
Zongming Fei, *University of Kentucky*
Guy Hembroff, *Michigan Tech University*
Frank Lin, *San Jose State University*
Tim Lin, *California Polytechnic University–Pomona*
Abdallah Shami, *University of Western Ontario*
Elsa Valeroso, *Eastern Michigan University*
Mark Weiser, *Oklahoma State University*
Ben Zhao, *University of California at Santa Barbara*

I acknowledge the invaluable contributions of professor Paul Amer for providing comments and feedbacks on the manuscript.

Special thanks go to the staff of McGraw-Hill. Raghu Srinivasan, the publisher, proved how a proficient publisher can make the impossible, possible. Melinda Bilecki, the developmental editor, gave help whenever I needed it. Joyce Watters, the project manager, guided me through the production process with enormous enthusiasm. I also thank Les Chappell of Macmillan Publishing Solutions in production, Laurie Janssen, the designer, and George F. Watson, the copy editor.

Behrouz A. Forouzan
January, 2009.

# *Trademarks*

Throughout the text I have used several trademarks. Rather than insert a trademark symbol with each mention of the trademark name, I acknowledge the trademarks here and state that they are used with no intention of infringing upon them. Other product names, trademarks, and registered trademarks are the property of their respective owners.

- Network File System and NFS are registered trademarks of Sun Microsystems, Inc.
- UNIX is a registered trademark of UNIX System Laboratories, Inc., a wholly owned subsidiary of Novell, Inc.
- Xerox is a trademark and Ethernet is a registered trademark of Xerox Corp.

# PART 1

# Introduction and Underlying Technologies

CHAPTER

# 1

# *Introduction*

The Internet is a structured, organized system. Before we discuss how it works and its relationship to TCP/IP, we first give a brief history of the Internet. We then define the concepts of protocols and standards and their relationships to each other. We discuss the various organizations that are involved in the development of Internet standards. These standards are not developed by any specific organization, but rather through a consensus of users. We discuss the mechanism through which these standards originated and matured. Also included in this introductory chapter is a section on Internet administrative groups.

## OBJECTIVES

*The chapter has several objectives:*

- To give a brief history of the Internet.
- To give the definition of the two often-used terms in the discussion of the Internet: *protocol* and *standard*.
- To categorize standard organizations involved in the Internet and give a brief discussion of each.
- To define Internet Standards and explain the mechanism through which these standards are developed.
- To discuss the Internet administration and give a brief description of each branch.

## 1.1 A BRIEF HISTORY

A **network** is a group of connected, communicating devices such as computers and printers. An internet (note the lowercase *i*) is two or more networks that can communicate with each other. The most notable internet is called the **Internet** (uppercase *I*), composed of hundreds of thousands of interconnected networks. Private individuals as well as various organizations such as government agencies, schools, research facilities, corporations, and libraries in more than 100 countries use the Internet. Millions of people are users. Yet this extraordinary communication system only came into being in 1969.

### ARPANET

In the mid-1960s, mainframe computers in research organizations were stand-alone devices. Computers from different manufacturers were unable to communicate with one another. The **Advanced Research Projects Agency (ARPA)** in the Department of Defense (DOD) was interested in finding a way to connect computers together so that the researchers they funded could share their findings, thereby reducing costs and eliminating duplication of effort.

In 1967, at an Association for Computing Machinery (ACM) meeting, ARPA presented its ideas for **ARPANET,** a small network of connected computers. The idea was that each host computer (not necessarily from the same manufacturer) would be attached to a specialized computer, called an *interface message processor* (IMP). The IMPs, in turn, would be connected to each other. Each IMP had to be able to communicate with other IMPs as well as with its own attached host.

By 1969, ARPANET was a reality. Four nodes, at the University of California at Los Angeles (UCLA), the University of California at Santa Barbara (UCSB), Stanford Research Institute (SRI), and the University of Utah, were connected via the IMPs to form a network. Software called the *Network Control Protocol* (NCP) provided communication between the hosts.

### Birth of the Internet

In 1972, Vint Cerf and Bob Kahn, both of whom were part of the core ARPANET group, collaborated on what they called the *Internetting Project*. They wanted to link different networks together so that a host on one network could communicate with a host on a second, different network. There were many problems to overcome: diverse packet sizes, diverse interfaces, and diverse transmission rates, as well as differing reliability requirements. Cerf and Kahn devised the idea of a device called a *gateway* to serve as the intermediary hardware to transfer data from one network to another.

## Transmission Control Protocol/Internetworking Protocol (TCP/IP)

Cerf and Kahn's landmark 1973 paper outlined the protocols to achieve end-to-end delivery of data. This was a new version of NCP. This paper on transmission control protocol (TCP) included concepts such as encapsulation, the datagram, and the functions of a gateway. A radical idea was the transfer of responsibility for error correction from the IMP to the host machine. This ARPA Internet now became the focus of the communication effort. Around this time responsibility for the ARPANET was handed over to the Defense Communication Agency (DCA).

In October 1977, an internet consisting of three different networks (ARPANET, packet radio, and packet satellite) was successfully demonstrated. Communication between networks was now possible.

Shortly thereafter, authorities made a decision to split TCP into two protocols: **Transmission Control Protocol (TCP)** and **Internet Protocol (IP).** IP would handle datagram routing while TCP would be responsible for higher level functions such as segmentation, reassembly, and error detection. The new combination became known as TCP/IP.

In 1981, under a DARPA contract, UC Berkeley modified the UNIX operating system to include TCP/IP. This inclusion of network software along with a popular operating system did much for the popularity of networking. The open (non-manufacturer-specific) implementation on Berkeley UNIX gave every manufacturer a working code base on which they could build their products.

In 1983, authorities abolished the original ARPANET protocols, and TCP/IP became the official protocol for the ARPANET. Those who wanted to use the Internet to access a computer on a different network had to be running TCP/IP.

## MILNET

In 1983, ARPANET split into two networks: **MILNET** for military users and ARPANET for nonmilitary users.

## CSNET

Another milestone in Internet history was the creation of CSNET in 1981. **CSNET** was a network sponsored by the National Science Foundation (NSF). The network was conceived by universities that were ineligible to join ARPANET due to an absence of defense ties to DARPA. CSNET was a less expensive network; there were no redundant links and the transmission rate was slower. It featured connections to ARPANET and Telenet, the first commercial packet data service.

By the middle 1980s, most U.S. universities with computer science departments were part of CSNET. Other institutions and companies were also forming their own networks and using TCP/IP to interconnect. The term *Internet*, originally associated with government-funded connected networks, now referred to the connected networks using TCP/IP protocols.

## NSFNET

With the success of CSNET, the NSF, in 1986, sponsored **NSFNET,** a backbone that connected five supercomputer centers located throughout the United States. Community

networks were allowed access to this backbone, a T-1 line with a 1.544-Mbps data rate, thus providing connectivity throughout the United States.

In 1990, ARPANET was officially retired and replaced by NSFNET. In 1995, NSFNET reverted back to its original concept of a research network.

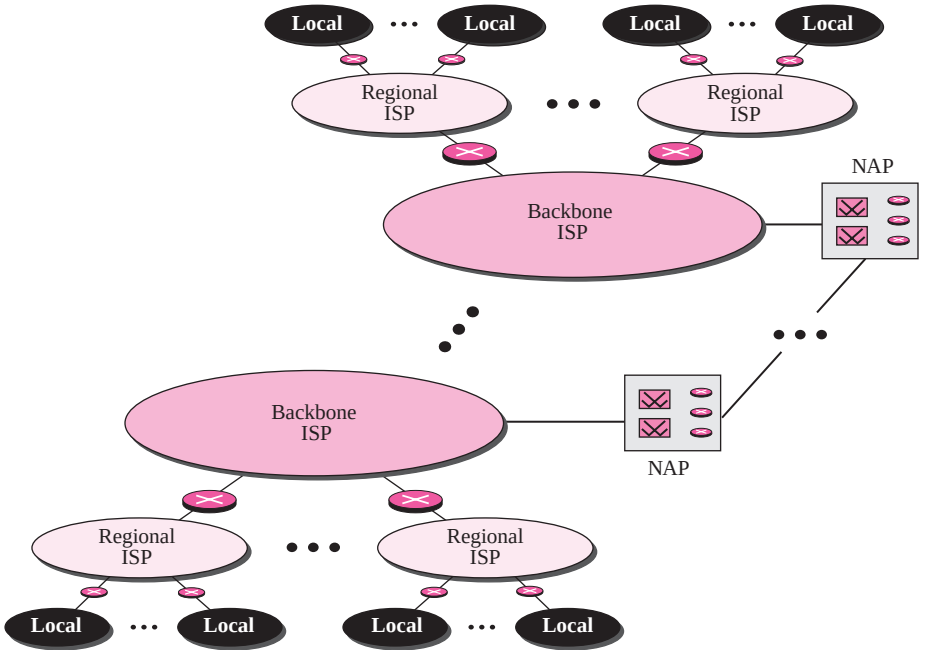
## ANSNET

In 1991, the U.S. government decided that NSFNET was not capable of supporting the rapidly increasing Internet traffic. Three companies, IBM, Merit, and MCI, filled the void by forming a nonprofit organization called Advanced Network and Services (ANS) to build a new, high-speed Internet backbone called **ANSNET.**

## The Internet Today

The Internet today is not a simple hierarchical structure. It is made up of many wide and local area networks joined by connecting devices and switching stations. It is difficult to give an accurate representation of the Internet because it is continuously changing—new networks are being added, existing networks need more addresses, and networks of defunct companies need to be removed. Today most end users who want Internet connection use the services of Internet service providers (ISPs). There are international service providers, national service providers, regional service providers, and local service providers. The Internet today is run by private companies, not the government. Figure 1.1 shows a conceptual (not geographical) view of the Internet.

**Figure 1.1** *Internet today*



### *Backbone ISPs*

Backbone ISPs are created and maintained by specialized companies. There are many backbone ISPs operating in North America; some of the most well-known are SprintLink, PSINet, UUNet Technology, AGIS, and internet MCI. To provide connectivity between the end users, these backbone networks are connected by complex switching stations (normally run by a third party) called **network access points (NAPs).** Some regional ISP networks are also connected to each other by private switching stations called peering points. Backbone ISPs normally operate at a high data rate (10 Gbps, for example).

### *Regional ISPs*

Regional ISPs are small ISPs that are connected to one or more backbone ISPs. They are at the second level of hierarchy with a lesser data rate.

### *Local ISPs*

Local ISPs provide direct service to the end users. The local ISPs can be connected to regional ISPs or directly to backbone ISPs. Most end users are connected to the local ISPs. Note that in this sense, a local ISP can be a company that just provides Internet services, a corporation with a network to supply services to its own employees, or a nonprofit organization, such as a college or a university, that runs its own network. Each of these can be connected to a regional or backbone service provider.

## World Wide Web

The 1990s saw the explosion of the Internet applications due to the emergence of the World Wide Web (WWW). The web was invented at CERN by Tim Berners-Lee. This invention has added the commercial applications to the Internet.

## Time Line

The following is a list of important Internet events in chronological order:

- **1969.** Four-node ARPANET established.
- **1970.** ARPA hosts implement NCP.
- **1973.** Development of TCP/IP suite begins.
- **1977.** An internet tested using TCP/IP.
- **1978.** UNIX distributed to academic/research sites.
- **1981.** CSNET established.
- **1983.** TCP/IP becomes the official protocol for ARPANET.
- **1983.** MILNET was born.
- **1986.** NSFNET established.
- **1990.** ARPANET decommissioned and replaced by NSFNET.
- **1995.** NSFNET goes back to being a research network.
- **1995.** Companies known as **Internet Service Providers (ISPs)** started.

### Growth of the Internet

The Internet has grown tremendously. In just a few decades, the number of networks has increased from tens to hundreds of thousands. Concurrently, the number of computers connected to the networks has grown from hundreds to hundreds of millions. The Internet is still growing. Factors that have an impact on this growth include the following:

- **New Protocols.** New protocols need to be added and deprecated ones need to be removed. For example, a protocol superior in many respects to IPv4 has been approved as a standard but is not yet fully implemented (see IPv6, Chapter 27).
- **New Technology.** New technologies are under development that will increase the capacity of networks and provide more bandwidth to the Internet's users.
- **Increasing Use of Multimedia.** It is predicted that the Internet, once just a vehicle to share data, will be used more and more for multimedia (audio and video).

## 1.2 PROTOCOLS AND STANDARDS

In this section, we define two widely used terms: protocols and standards. First, we define *protocol*, which is synonymous with "rule." Then we discuss *standards*, which are agreed-upon rules.

### Protocols

Communication between two people or two devices needs to follow some protocol. A **protocol** is a set of rules that governs communication. For example, in a face-to-face communication between two persons, there is a set of implicit rules in each culture that define how two persons should start the communication, how to continue the communication, and how to end the communication. Similarly, in a telephone conversation, there are a set of rules that we need to follow. There is a rule how to make connection (dialing the telephone number), how to respond to the call (picking up the receiver), how to greet, how to let the communication flow smoothly by listening when the other party is talking, and finally how to end the communication (hanging up).

In computer networks, communication occurs between entities in different systems. An entity is anything capable of sending or receiving information. However, two entities cannot simply send bit streams to each other and expect to be understood. For communication to occur, the entities must agree on a protocol. A protocol defines what is communicated, how it is communicated, and when it is communicated. The key elements of a protocol are syntax, semantics, and timing.

- **Syntax.** Syntax refers to the structure or format of the data, meaning the order in which they are presented. For example, a simple protocol might expect the first 8 bits of data to be the address of the sender, the second 8 bits to be the address of the receiver, and the rest of the stream to be the message itself. The data order is also applied to the order of bits when they are stored or transmitted. Different computers may store data in different bit orders. When these computers communicate, this difference needs to be resolved.

- **Semantics.** Semantics refers to the meaning of each section of bits. How is a particular pattern to be interpreted, and what action is to be taken based on that interpretation? For example, does an address identify the route to be taken or the final destination of the message?
- **Timing.** Timing refers to two characteristics: when data should be sent and how fast it can be sent. For example, if a sender produces data at 100 megabits per second (100 Mbps) but the receiver can process data at only 1 Mbps, the transmission will overload the receiver and data will be largely lost.

### Standards

Standards are essential in creating and maintaining an open and competitive market for equipment manufacturers and also in guaranteeing national and international interoperability of data and telecommunications technology and processes. They provide guidelines to manufacturers, vendors, government agencies, and other service providers to ensure the kind of interconnectivity necessary in today's marketplace and in international communications.

Data communication standards fall into two categories: *de facto* (meaning "by fact" or "by convention") and *de jure* (meaning "by law" or "by regulation").

- **De facto.** Standards that have not been approved by an organized body but have been adopted as standards through widespread use are **de facto standards.** De facto standards are often established originally by manufacturers that seek to define the functionality of a new product or technology. Examples of de facto standards are MS Office and various DVD standards.
- **De jure. De jure standards** are those that have been legislated by an officially recognized body.

## 1.3 STANDARDS ORGANIZATIONS

Standards are developed through the cooperation of standards creation committees, forums, and government regulatory agencies.

### Standards Creation Committees

While many organizations are dedicated to the establishment of standards, data communications in North America rely primarily on those published by the following:

- **International Standards Organization (ISO).** The International Standards Organization (ISO; also referred to as the International Organization for Standardization) is a multinational body whose membership is drawn mainly from the standards creation committees of various governments throughout the world. Created in 1947, the ISO is an entirely voluntary organization dedicated to worldwide agreement on international standards. With a membership that currently includes representative bodies from many industrialized nations, it aims to facilitate the international exchange of goods and services by providing models for compatibility, improved quality, increased productivity, and decreased prices. The ISO is active

in developing cooperation in the realms of scientific, technological, and economic activity. Of primary concern to this book are the ISO's efforts in the field of information technology, which have resulted in the creation of the Open Systems Interconnection (OSI) model for network communications. The United States is represented in the ISO by ANSI.

- **International Telecommunications Union–Telecommunications Standards Sector (ITU-T).** By the early 1970s, a number of countries were defining national standards for telecommunications, but there was still little international compatibility. The United Nations responded by forming, as part of its International Telecommunications Union (ITU), a committee, the **Consultative Committee for International Telegraphy and Telephony (CCITT).** This committee was devoted to the research and establishment of standards for telecommunications in general and phone and data systems in particular. On March 1, 1993, the name of this committee was changed to the International Telecommunications Union–Telecommunications Standards Sector (ITU-T).
- **American National Standards Institute (ANSI).** Despite its name, the American National Standards Institute (ANSI) is a completely private, nonprofit corporation not affiliated with the U.S. federal government. However, all ANSI activities are undertaken with the welfare of the United States and its citizens occupying primary importance. ANSI's expressed aims include serving as the national coordinating institution for voluntary standardization in the United States, furthering the adoption of standards as a way of advancing the U.S. economy, and ensuring the participation and protection of the public interests. ANSI members include professional societies, industry associations, governmental and regulatory bodies, and consumer groups.
- **Institute of Electrical and Electronics Engineers (IEEE).** The Institute of Electrical and Electronics Engineers (IEEE) is the largest professional engineering society in the world. International in scope, it aims to advance theory, creativity, and product quality in the fields of electrical engineering, electronics, and radio as well as in all related branches of engineering. As one of its goals, the IEEE oversees the development and adoption of international standards for computing and communication.
- **Electronic Industries Association (EIA).** Aligned with ANSI, the Electronic Industries Association (EIA) is a nonprofit organization devoted to the promotion of electronics manufacturing concerns. Its activities include public awareness education and lobbying efforts in addition to standards development. In the field of information technology, the EIA has made significant contributions by defining physical connection interfaces and electronic signaling specifications for data communications.
- **World Wide Web Consortium (W3C).** Tim Berners-Lee founded this consortium at Massachusetts Institutue of Technology Laboratory for Computer Science. It was founded to provide computability in industry for new standards. W3C has created regional offices around the world.
- **Open Mobile Alliance (OMA).** The standards organizaion OMA was created to gather different forums in computer networking and wireless technology under the umbrella of one single authority. Its mission is to provide unified standards for application protocols.

### Forums

Telecommunications technology development is moving faster than the ability of standards committees to ratify standards. Standards committees are procedural bodies and by nature slow moving. To accommodate the need for working models and agreements and to facilitate the standardization process, many special-interest groups have developed *forums* made up of representatives from interested corporations. The forums work with universities and users to test, evaluate, and standardize new technologies. By concentrating their efforts on a particular technology, the forums are able to speed acceptance and use of those technologies in the telecommunications community. The forums present their conclusions to the standards bodies. Some important forums for the telecommunications industry include the following:

- **Frame Relay Forum.** The Frame Relay Forum was formed by Digital Equipment Corporation, Northern Telecom, Cisco, and StrataCom to promote the acceptance and implementation of Frame Relay. Today, it has around 40 members representing North America, Europe, and the Pacific Rim. Issues under review include flow control, encapsulation, translation, and multicasting. The forum's results are submitted to the ISO.
- **ATM Forum.** The ATM Forum promotes the acceptance and use of Asynchronous Transfer Mode (ATM) technology. The ATM Forum is made up of customer premises equipment (e.g., PBX systems) vendors and central office (e.g., telephone exchange) providers. It is concerned with the standardization of services to ensure interoperability.
- **Universal Plug and Play (UPnP) Forum.** The UPnP forum is a computer network forum that supports and promotes simplifying the implementation of networks by creating zero-configuration networking devices. A UPnP-compatible device can join a network without any configuration.

### Regulatory Agencies

All communications technology is subject to regulation by government agencies such as the Federal Communications Commission in the United States. The purpose of these agencies is to protect the public interest by regulating radio, television, and wire/cable communications.

- **Federal Communications Commission (FCC).** The Federal Communications Commission (FCC) has authority over interstate and international commerce as it relates to communications.

**The websites for the above organizations are given in Appendix G.**

## 1.4 INTERNET STANDARDS

An **Internet standard** is a thoroughly tested specification that is useful to and adhered to by those who work with the Internet. It is a formalized regulation that must be followed. There is a strict procedure by which a specification attains Internet standard status. A specification begins as an Internet draft. An **Internet draft** is a working document (a work in progress) with no official status and a six-month lifetime. Upon recommendation from the

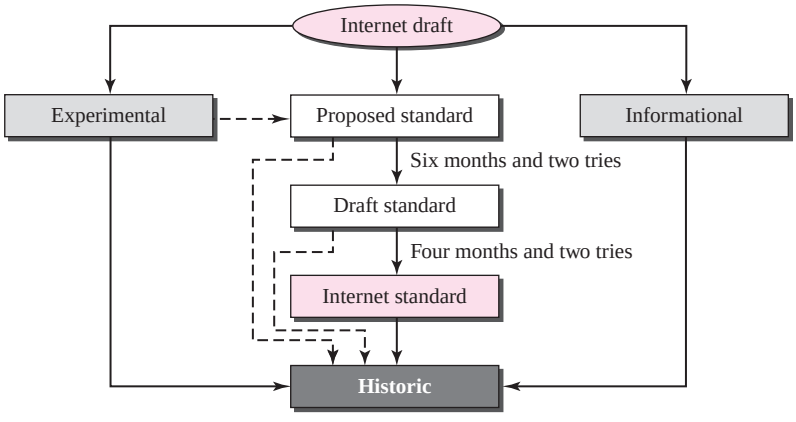
Internet authorities, a draft may be published as a **Request for Comment (RFC).** Each RFC is edited, assigned a number, and made available to all interested parties.

RFCs go through maturity levels and are categorized according to their requirement level.

## Maturity Levels

An RFC, during its lifetime, falls into one of six **maturity levels:** proposed standard, draft standard, Internet standard, historic, experimental, and informational (see Figure 1.2).

**Figure 1.2** *Maturity levels of an RFC*




### *Proposed Standard*

A proposed standard is a specification that is stable, well understood, and of sufficient interest to the Internet community. At this level, the specification is usually tested and implemented by several different groups.

### *Draft Standard*

A proposed standard is elevated to draft standard status after at least two successful independent and interoperable implementations. Barring difficulties, a draft standard, with modifications if specific problems are encountered, normally becomes an Internet standard.

### *Internet Standard*

A draft standard reaches Internet standard status after demonstrations of successful implementation.

### *Historic*

The historic RFCs are significant from a historical perspective. They either have been superseded by later specifications or have never passed the necessary maturity levels to become an Internet standard.

### *Experimental*

An RFC classified as experimental describes work related to an experimental situation that does not affect the operation of the Internet. Such an RFC should not be implemented in any functional Internet service.

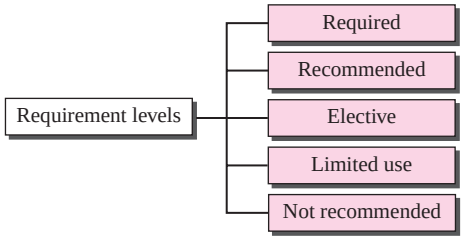
### *Informational*

An RFC classified as informational contains general, historical, or tutorial information related to the Internet. It is usually written by someone in a non-Internet organization, such as a vendor.

## Requirement Levels

RFCs are classified into five **requirement levels:** required, recommended, elective, limited use, and not recommended (see Figure 1.3).

**Figure 1.3** *Requirement levels of an RFC*




### *Required*

An RFC is labeled *required* if it must be implemented by all Internet systems to achieve minimum conformance. For example, IP (Chapter 7) and ICMP (Chapter 9) are required protocols.

### *Recommended*

An RFC labeled *recommended* is not required for minimum conformance; it is recommended because of its usefulness. For example, FTP (Chapter 21) and TELNET (Chapter 20) are recommended protocols.

### *Elective*

An RFC labeled *elective* is not required and not recommended. However, a system can use it for its own benefit.

### *Limited Use*

An RFC labeled *limited use* should be used only in limited situations. Most of the experimental RFCs fall under this category.

### *Not Recommended*

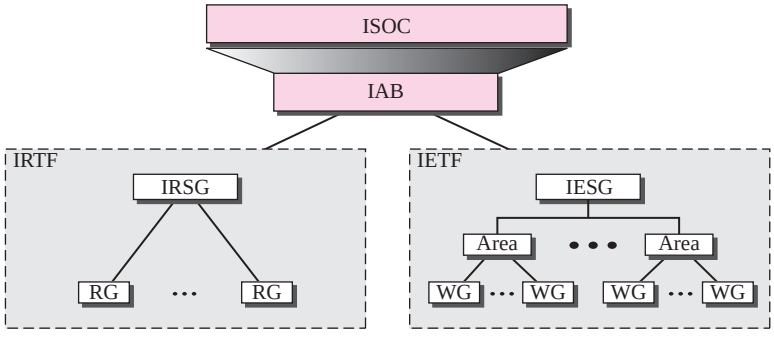
An RFC labeled *not recommended* is inappropriate for general use. Normally a historic (deprecated) RFC may fall under this category.

**RFCs can be found at http://www.rfc-editor.org.**

## 1.5 INTERNET ADMINISTRATION

The Internet, with its roots primarily in the research domain, has evolved and gained a broader user base with significant commercial activity. Various groups that coordinate Internet issues have guided this growth and development. Appendix G gives the addresses, e-mail addresses, and telephone numbers for some of these groups. Figure 1.4 shows the general organization of Internet administration.

**Figure 1.4** *Internet administration*




### Internet Society (ISOC)

The **Internet Society (ISOC)** is an international, nonprofit organization formed in 1992 to provide support for the Internet standards process. ISOC accomplishes this through maintaining and supporting other Internet administrative bodies such as IAB, IETF, IRTF, and IANA (see the following sections). ISOC also promotes research and other scholarly activities relating to the Internet.

### Internet Architecture Board (IAB)

The **Internet Architecture Board (IAB)** is the technical advisor to the ISOC. The main purposes of the IAB are to oversee the continuing development of the TCP/IP Protocol Suite and to serve in a technical advisory capacity to research members of the Internet community. IAB accomplishes this through its two primary components, the Internet Engineering Task Force (IETF) and the Internet Research Task Force (IRTF). Another responsibility of the IAB is the editorial management of the RFCs, described earlier in this chapter. IAB is also the external liaison between the Internet and other standards organizations and forums.

### Internet Engineering Task Force (IETF)

The **Internet Engineering Task Force (IETF)** is a forum of working groups managed by the Internet Engineering Steering Group (IESG). IETF is responsible for identifying operational problems and proposing solutions to these problems. IETF

also develops and reviews specifications intended as Internet standards. The working groups are collected into areas, and each area concentrates on a specific topic. Currently nine areas have been defined, although this is by no means a hard and fast number. The areas are:

- Applications
- Internet protocols
- Routing
- Operations
- User services
- Network management
- Transport
- Internet protocol next generation (IPng)
- Security

### Internet Research Task Force (IRTF)

The **Internet Research Task Force (IRTF)** is a forum of working groups managed by the Internet Research Steering Group (IRSG). IRTF focuses on long-term research topics related to Internet protocols, applications, architecture, and technology.

### Internet Assigned Numbers Authority (IANA) and Internet Corporation for Assigned Names and Numbers (ICANN)

The **Internet Assigned Numbers Authority (IANA),** supported by the U.S. government, was responsible for the management of Internet domain names and addresses until October 1998. At that time the **Internet Corporation for Assigned Names and Numbers (ICANN),** a private nonprofit corporation managed by an international board, assumed IANA operations.

### Network Information Center (NIC)

The **Network Information Center (NIC)** is responsible for collecting and distributing information about TCP/IP protocols.

**The addresses and websites for Internet organizations can be found in Appendix G.**

## 1.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and websites. The items enclosed in brackets refer to the reference list at the end of the book.

### Books and Papers

Several books and papers give an easy but thorough coverage of Internet history including [Seg 98], [Lei et al. 98], [Kle 04], [Cer 89], and [Jen et al. 86].

### Websites

The following websites give more information about topics discussed in this chapter.

| | |
|---|---|
| ietf.org | The site of IETF |
| w3c.org | The site of W3C standard organization |

## 1.7 KEY TERMS

Advanced Research Projects Agency (ARPA)
American National Standards Institute (ANSI)
ANSNET
ARPANET
ATM Forum
Consultative Committee for International Telegraphy and Telephony (CCITT)
CSNET
de facto standards
de jure standards
Electronic Industries Association (EIA)
Federal Communications Commission (FCC)
Frame Relay Forum
Institute of Electrical and Electronics Engineers (IEEE)
International Standards Organization (ISO)
International Telecommunications Union–Telecommunications Standards Sector (ITUT)
Internet Architecture Board (IAB)
Internet Assigned Numbers Authority (IANA)
Internet Corporation for Assigned Names and Numbers (ICANN)
Internet draft
Internet Engineering Task Force (IETF)
Internet Research Task Force (IRTF)
Internet Service Provider (ISP)
Internet Society (ISOC)
Internet standard
Internet Protocol (IP)
maturity levels
MILNET
network
network access points (NAPs)
Network Information Center (NIC)
NSFNET
protocol
Open Mobile Alliance (OMA)
Request for Comment (RFC)
requirement levels
semantics
syntax
timing
Transmission Control Protocol (TCP)
Universal Plug and Play (UPnP) Forum
World Wide Web Consortium (W3C)

## 1.8 SUMMARY

- A network is a group of connected, communicating devices. An internet is two or more networks that can communicate with each other. The most notable internet is called the Internet, composed of hundreds of thousands of interconnected networks.
- The history of internetworking started with ARPA in the mid-1960s. The birth of the Internet can be associated with the work of Cerf and Kahn and the invention

of a gateway to connect networks. In 1977, the Defense Communication Agency (DCA) took the responsibility of the ARPANET and used two protocols called TCP and IP to handle the routing of datagrams between individual networks. MILNET, CSNET, NSFNET, ANSNET, are all evolved from the ARPANET.

- The Internet today is made up of many wide and local area networks joined by connecting devices and switching stations. Today most end users who want Internet connection use the services of Internet service providers (ISPs). There are backbone ISPs, regional ISPs, and local ISPs.
- A protocol is a set of rules that governs communication. The key elements of a protocol are syntax, semantics, and timing. In computer networks, communication occurs between entities in different systems. For communication to occur, the entities must agree on a protocol. A protocol defines what is communicated, how it is communicated, and when it is communicated.
- Standards are essential in creating and maintaining an open and competitive market. They provide guidelines to manufacturers, vendors, government agencies, and other service providers to ensure the kind of interconnectivity necessary in today's marketplace and in international communications. Data communication standards fall into two categories: *de facto* and *de jure*.
- An Internet standard is a thoroughly tested specification that is useful to and adhered to by those who work with the Internet. An Internet draft is a working document (a work in progress) with no official status and a six-month lifetime. Upon recommendation from the Internet authorities, a draft may be published as a Request for Comment (RFC). Each RFC is edited, assigned a number, and made available to all interested parties. RFCs go through maturity levels and are categorized according to their requirement level.
- The Internet administration has evolved with the Internet. ISOC promotes research and activities. IAB is the technical advisor to the ISOC. IETF is a forum of working groups responsible for operational problems. IRTF is a forum of working groups focusing on long-term research topics. ICANN is responsible for the management of Internet domain names and addresses. NIC is responsible for collecting and distributing information about TCP/IP protocols.

## 1.9 PRACTICE SET

### Exercises

1. Use the Internet to find the number of RFCs.
2. Use the Internet to find the subject matter of RFCs 2418 and 1603.
3. Use the Internet to find the RFC that discusses the IRTF working group guidelines and procedures.
4. Use the Internet to find two examples of historic RFCs.
5. Use the Internet to find two examples of experimental RFCs.

6. Use the Internet to find two examples of informational RFCs.
7. Use the Internet to find the RFC that discusses the FTP application.
8. Use the Internet to find the RFC for the Internet Protocol (IP).
9. Use the Internet to find the RFC for the Transmission Control Protocol (TCP).
10. Use the Internet to find the RFC that details the Internet standards process.

## Research Activities

11. Research and find three standards developed by ITU-T.
12. Research and find three standards developed by ANSI.
13. EIA has developed some standards for interfaces. Research and find two of these standards. What is EIA 232?
14. Research and find three regulations devised by FCC concerning AM and FM transmission.

CHAPTER

# 2

# *The OSI Model and the TCP/IP Protocol Suite*

The layered model that dominated data communication and networking literature before 1990 was the **Open Systems Interconnection (OSI) model.** Everyone believed that the OSI model would become the ultimate standard for data communications—but this did not happen. The **TCP/IP protocol suite** became the dominant commercial architecture because it was used and tested extensively in the Internet; the OSI model was never fully implemented.

In this chapter, we first briefly discuss the OSI model and then we concentrate on TCP/IP as a protocol suite.

## OBJECTIVES

*The chapter has several objectives:*

- To discuss the idea of multiple layering in data communication and networking and the interrelationship between layers.
- To discuss the OSI model and its layer architecture and to show the interface between the layers.
- To briefly discuss the functions of each layer in the OSI model.
- To introduce the TCP/IP protocol suite and compare its layers with the ones in the OSI model.
- To show the functionality of each layer in the TCP/IP protocol with some examples.
- To discuss the addressing mechanism used in some layers of the TCP/IP protocol suite for the delivery of a message from the source to the destination.

## 2.1 PROTOCOL LAYERS

In Chapter 1, we discussed that a protocol is required when two entities need to communicate. When communication is not simple, we may divide the complex task of communication into several layers. In this case, we may need several protocols, one for each layer.

Let us use a scenario in communication in which the role of protocol layering may be better understood. We use two examples. In the first example, communication is so simple that it can occur in only one layer. In the second example, we need three layers.

### Example 2.1

Assume Maria and Ann are neighbors with a lot of common ideas. However, Maria speaks only Spanish, and Ann speaks only English. Since both have learned the sign language in their childhood, they enjoy meeting in a cafe a couple of days per week and exchange their ideas using signs. Occasionally, they also use a bilingual dictionary. Communication is face to face and happens in one layer as shown in Figure 2.1.

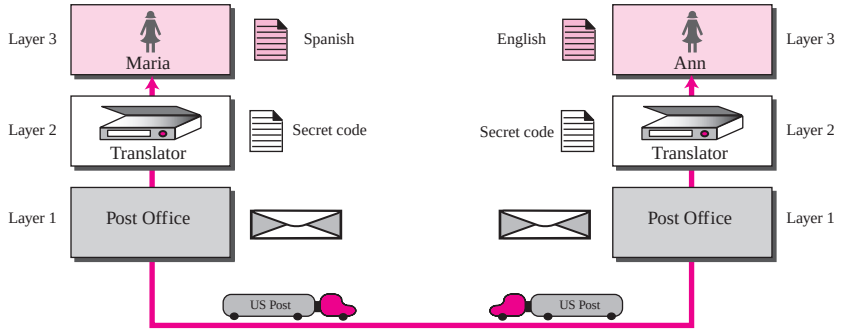
**Figure 2.1** *Example 2.1*


### Example 2.2

Now assume that Ann has to move to another town because of her job. Before she moves, the two meet for the last time in the same cafe. Although both are sad, Maria surprises Ann when she opens a packet that contains two small machines. The first machine can scan and transform a letter in English to a secret code or vice versa. The other machine can scan and translate a letter in Spanish to the same secret code or vice versa. Ann takes the first machine; Maria keeps the second one. The two friends can still communicate using the secret code, as shown in Figure 2.2.

Communication between Maria and Ann happens as follows. At the third layer, Maria writes a letter in Spanish, the language she is comfortable with. She then uses the translator machine that scans the letter and creates a letter in the secret code. Maria then puts the letter in an envelop and drops it to the post office box. The letter is carried by the post office truck to the post office of the city where Ann lives now. In the post office, the letter is delivered to the Ann residence. Ann uses her own machine to change the secret code to a letter in the English language. The communication from Ann to Maria uses the same process, but in the reverse direction. The communication in both directions is carried in the secret code, a language that neither Maria nor Ann understands, but through the layered communication, they can exchange ideas.

**Figure 2.2** *Example 2.2*




### Hierarchy

Using Example 2.2, there are three different activities at the sender site and another three activities at the receiver site. The task of transporting the letter between the sender and the receiver is done by the carrier. Something that is not obvious immediately is that the tasks must be done in the order given in the hierarchy. At the sender site, the letter must be written, translated to secret code, and dropped in the mailbox before being picked up by the letter carrier and delivered to the post office. At the receiver site, the letter must be dropped in the recipient mailbox before being picked up and read by the recipient.

### Services

Each layer at the sending site uses the services of the layer immediately below it. The sender at the higher layer uses the services of the middle layer. The middle layer uses the services of the lower layer. The lower layer uses the services of the carrier.

## 2.2 THE OSI MODEL

Established in 1947, the **International Standards Organization (ISO)** is a multinational body dedicated to worldwide agreement on international standards. Almost three-fourths of countries in the world are represented in the ISO. An ISO standard that covers all aspects of network communications is the Open Systems Interconnection (OSI) model. It was first introduced in the late 1970s.

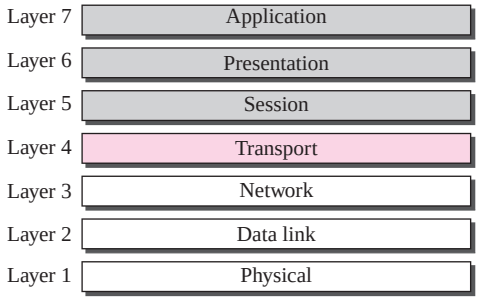
**ISO is the organization; OSI is the model.**

An **open system** is a set of protocols that allows any two different systems to communicate regardless of their underlying architecture. The purpose of the OSI model is

to show how to facilitate communication between different systems without requiring changes to the logic of the underlying hardware and software. The OSI model is not a protocol; it is a model for understanding and designing a network architecture that is flexible, robust, and interoperable. The OSI model was intended to be the basis for the creation of the protocols in the OSI stack.

The OSI model is a layered framework for the design of network systems that allows communication between all types of computer systems. It consists of seven separate but related layers, each of which defines a part of the process of moving information across a network (see Figure 2.3). Understanding the fundamentals of the OSI model provides a solid basis for exploring data communications.

**Figure 2.3** *The OSI model*



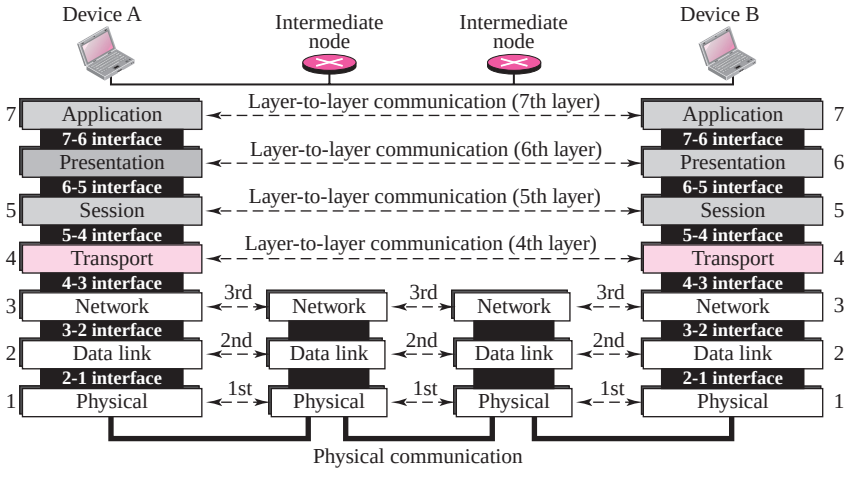

## Layered Architecture

The OSI model is composed of seven ordered layers: physical (layer 1), data link (layer 2), network (layer 3), transport (layer 4), session (layer 5), presentation (layer 6), and application (layer 7). Figure 2.4 shows the layers involved when a message is sent from device A to device B. As the message travels from A to B, it may pass through many intermediate nodes. These intermediate nodes usually involve only the first three layers of the OSI model.

In developing the model, the designers distilled the process of transmitting data to its most fundamental elements. They identified which networking functions had related uses and collected those functions into discrete groups that became the layers. Each layer defines a family of functions distinct from those of the other layers. By defining and localizing functionality in this fashion, the designers created an architecture that is both comprehensive and flexible. Most important, the OSI model allows complete interoperability between otherwise incompatible systems.

Within a single machine, each layer calls upon the services of the layer just below it. Layer 3, for example, uses the services provided by layer 2 and provides services for layer 4. Between machines, layer *x* on one machine logically communicates with layer *x* on another machine. This communication is governed by an agreed-upon series of rules and conventions called protocols.

**Figure 2.4** *OSI layers*




## Layer-to-Layer Communication

In Figure 2.4, device A sends a message to device B (through intermediate nodes). At the sending site, the message is moved down from layer 7 to layer 1. At layer 1 the entire package is converted to a form that can be transferred to the receiving site. At the receiving site, the message is moved up from layer 1 to layer 7.

### *Interfaces between Layers*

The passing of the data and network information down through the layers of the sending device and back up through the layers of the receiving device is made possible by an **interface** between each pair of adjacent layers. Each interface defines what information and services a layer must provide for the layer above it. Well-defined interfaces and layer functions provide modularity to a network. As long as a layer provides the expected services to the layer above it, the specific implementation of its functions can be modified or replaced without requiring changes to the surrounding layers.

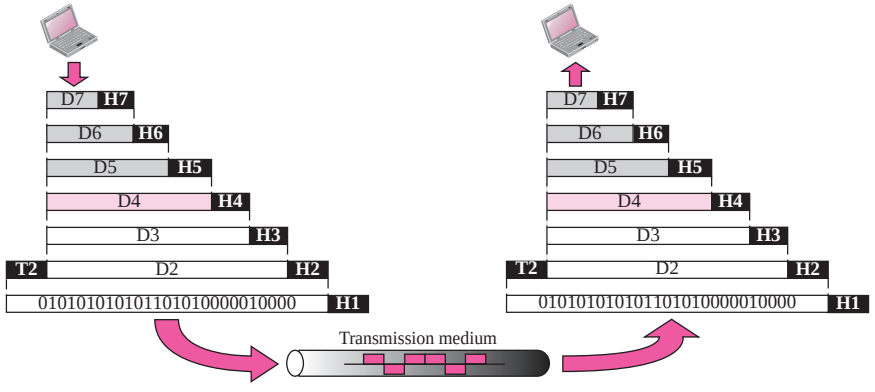
### *Organization of the Layers*

The seven layers can be thought of as belonging to three subgroups. Layers 1, 2, and 3—physical, data link, and network—are the network support layers; they deal with the physical aspects of moving data from one device to another (such as electrical specifications, physical connections, physical addressing, and transport timing and reliability). Layers 5, 6, and 7—session, presentation, and application—can be thought of as the user support layers; they allow interoperability among unrelated software systems. Layer 4, the transport layer, links the two subgroups and ensures that what the lower layers have transmitted is in a form that the upper layers can use.

The upper OSI layers are almost always implemented in software; lower layers are a combination of hardware and software, except for the physical layer, which is mostly hardware.

In Figure 2.5, which gives an overall view of the OSI layers, D7 data means the data unit at layer 7, D6 data means the data unit at layer 6, and so on. The process starts at layer 7 (the application layer), then moves from layer to layer in descending, sequential order. At each layer, a header can be added to the data unit. At layer 2, a trailer may also be added. When the formatted data unit passes through the physical layer (layer 1), it is changed into an electromagnetic signal and transported along a physical link.

**Figure 2.5** *An exchange using the OSI model*




Upon reaching its destination, the signal passes into layer 1 and is transformed back into digital form. The data units then move back up through the OSI layers. As each block of data reaches the next higher layer, the headers and trailers attached to it at the corresponding sending layer are removed, and actions appropriate to that layer are taken. By the time it reaches layer 7, the message is again in a form appropriate to the application and is made available to the recipient.

## Encapsulation

Figure 2.5 reveals another aspect of data communications in the OSI model: encapsulation. A packet at level 7 is encapsulated in the packet at level 6. The whole packet at level 6 is encapsulated in a packet at level 5, and so on.

In other words, the data part of a packet at level $N$ is carrying the whole packet (data and overhead) from level $N + 1$. The concept is called encapsulation because level $N$ is not aware what part of the encapsulated packet is data and what part is the header or trailer. For level $N$, the whole packet coming from level $N + 1$ is treated as one integral unit.

## Layers in the OSI Model

In this section we briefly describe the functions of each layer in the OSI model.

### *Physical Layer*

The **physical layer** coordinates the functions required to carry a bit stream over a physical medium. It deals with the mechanical and electrical specifications of the interface and transmission media. It also defines the procedures and functions that physical devices and interfaces have to perform for transmission to occur.

**The physical layer is responsible for moving individual bits from one (node) to the next.**

The physical layer is also concerned with the following:

- **Physical characteristics of interfaces and media.** The physical layer defines the characteristics of the interface between the devices and the transmission media. It also defines the type of transmission media (see Chapter 3).
- **Representation of bits.** The physical layer data consists of a stream of **bits** (sequence of 0s or 1s) with no interpretation. To be transmitted, bits must be encoded into signals—electrical or optical. The physical layer defines the type of **encoding** (how 0s and 1s are changed to signals).
- **Data rate.** The **transmission rate**—the number of bits sent each second—is also defined by the physical layer. In other words, the physical layer defines the duration of a bit, which is how long it lasts.
- **Synchronization of bits.** The sender and receiver must not only use the same bit rate but must also be synchronized at the bit level. In other words, the sender and the receiver clocks must be synchronized.
- **Line configuration.** The physical layer is concerned with the connection of devices to the media. In a **point-to-point configuration,** two devices are connected together through a dedicated link. In a **multipoint configuration,** a link is shared between several devices.
- **Physical topology.** The physical topology defines how devices are connected to make a network. Devices can be connected using a **mesh topology** (every device connected to every other device), a **star topology** (devices are connected through a central device), a **ring topology** (each device is connected to the next, forming a ring), or a **bus topology** (every device on a common link).
- **Transmission mode.** The physical layer also defines the direction of transmission between two devices: simplex, half-duplex, or full-duplex. In the **simplex mode,** only one device can send; the other can only receive. The simplex mode is a one-way communication. In the **half-duplex mode,** two devices can send and receive, but not at the same time. In a **full-duplex** (or simply duplex) **mode,** two devices can send and receive at the same time.

### *Data Link Layer*

The **data link layer** transforms the physical layer, a raw transmission facility, to a reliable link. It makes the physical layer appear error-free to the upper layer (network layer). Other responsibilities of the data link layer include the following:

- **Framing.** The data link layer divides the stream of bits received from the network layer into manageable data units called **frames.**
- **Physical addressing.** If frames are to be distributed to different systems on the network, the data link layer adds a header to the frame to define the sender and/or receiver of the frame. If the frame is intended for a system outside the sender's network, the receiver address is the address of the connecting device that connects the network to the next one.
- **Flow control.** If the rate at which the data is absorbed by the receiver is less than the rate produced at the sender, the data link layer imposes a flow control mechanism to prevent overwhelming the receiver.
- **Error control.** The data link layer adds reliability to the physical layer by adding mechanisms to detect and retransmit damaged or lost frames. It also uses a mechanism to recognize duplicate frames. Error control is normally achieved through a trailer added to the end of the frame.
- **Access control.** When two or more devices are connected to the same link, data link layer protocols are necessary to determine which device has control over the link at any given time.

### *Network Layer*

The **network layer** is responsible for the source-to-destination delivery of a packet, possibly across multiple networks (links). Whereas the data link layer oversees the delivery of the packet between two systems on the same network (link), the network layer ensures that each packet gets from its point of origin to its final destination.

If two systems are connected to the same link, there is usually no need for a network layer. However, if the two systems are attached to different networks (links) with connecting devices between the networks (links), there is often a need for the network layer to accomplish source-to-destination delivery. Other responsibilities of the network layer include the following:

- **Logical addressing.** The physical addressing implemented by the data link layer handles the addressing problem locally. If a packet passes the network boundary, we need another addressing system to help distinguish the source and destination systems. The network layer adds a header to the packet coming from the upper layer that, among other things, includes the logical addresses of the sender and receiver.
- **Routing.** When independent networks or links are connected together to create **internetworks** (network of networks) or a large network, the connecting devices (called *routers* or *switches*) route or switch the packets to their final destination. One of the functions of the network layer is to provide this mechanism.

### *Transport Layer*

The **transport layer** is responsible for **process-to-process delivery** of the entire message. A process is an application program running on the host. Whereas the network layer oversees **source-to-destination delivery** of individual packets, it does not recognize any relationship between those packets. It treats each one independently, as though each piece belonged to a separate message, whether or not it does. The transport layer, on the other hand, ensures that the whole message arrives intact and in order, overseeing both error control and flow control at the source-to-destination level. Other responsibilities of the transport layer include the following:

- **Service-point addressing.** Computers often run several programs at the same time. For this reason, source-to-destination delivery means delivery not only from one computer to the next but also from a specific process (running program) on one computer to a specific process (running program) on the other. The transport layer header must therefore include a type of address called a *service-point address* (or port address). The network layer gets each packet to the correct computer; the transport layer gets the entire message to the correct process on that computer.
- **Segmentation and reassembly.** A message is divided into transmittable segments, with each segment containing a sequence number. These numbers enable the transport layer to reassemble the message correctly upon arriving at the destination and to identify and replace packets that were lost in transmission.
- **Connection control.** The transport layer can be either connectionless or connection-oriented. A connectionless transport layer treats each segment as an independent packet and delivers it to the transport layer at the destination machine. A connection-oriented transport layer makes a connection with the transport layer at the destination machine first before delivering the packets. After all the data are transferred, the connection is terminated.
- **Flow control.** Like the data link layer, the transport layer is responsible for flow control. However, flow control at this layer is performed end to end rather than across a single link.
- **Error control.** Like the data link layer, the transport layer is responsible for error control. However, error control at this layer is performed process-to-process rather than across a single link. The sending transport layer makes sure that the entire message arrives at the receiving transport layer without *error* (damage, loss, or duplication). Error correction is usually achieved through retransmission.

### *Session Layer*

The services provided by the first four layers (physical, data link, network and transport) are not sufficient for some processes. The **session layer** is the network *dialog controller.* It establishes, maintains, and synchronizes the interaction between communicating systems. Specific responsibilities of the session layer include the following:

- **Dialog control.** The session layer allows two systems to enter into a dialog. It allows the communication between two processes to take place in either half-duplex (one way at a time) or full-duplex (two ways at a time) mode.

- **Synchronization.** The session layer allows a process to add checkpoints **(synchronization points)** into a stream of data. For example, if a system is sending a file of 2,000 pages, it is advisable to insert checkpoints after every 100 pages to ensure that each 100-page unit is received and acknowledged independently. In this case, if a crash happens during the transmission of page 523, the only pages that need to be resent after system recovery are pages 501 to 523. Pages previous to 501 need not be resent.

### *Presentation Layer*

The **presentation layer** is concerned with the syntax and semantics of the information exchanged between two systems. Specific responsibilities of the presentation layer include the following:

- **Translation.** The processes (running programs) in two systems are usually exchanging information in the form of character strings, numbers, and so on. The information should be changed to bit streams before being transmitted. Because different computers use different encoding systems, the presentation layer is responsible for interoperability between these different encoding methods. The presentation layer at the sender changes the information from its sender-dependent format into a common format. The presentation layer at the receiving machine changes the common format into its receiver-dependent format.
- **Encryption.** To carry sensitive information a system must be able to assure privacy. Encryption means that the sender transforms the original information to another form and sends the resulting message out over the network. Decryption reverses the original process to transform the message back to its original form.
- **Compression.** Data compression reduces the number of bits contained in the information. Data compression becomes particularly important in the transmission of multimedia such as text, audio, and video.

### *Application Layer*

The **application layer** enables the user, whether human or software, to access the network. It provides user interfaces and support for services such as electronic mail, remote file access and transfer, shared database management, and other types of distributed information services. Specific services provided by the application layer include the following:

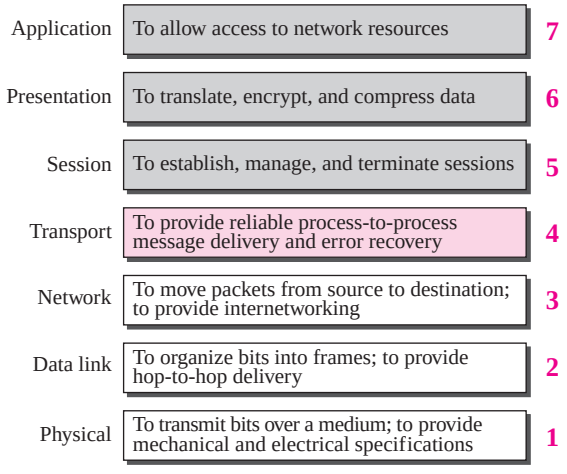
- **Network virtual terminal.** A network virtual terminal is a software version of a physical terminal and allows a user to log on to a remote host. To do so, the application creates a software emulation of a terminal at the remote host. The user's computer talks to the software terminal, which, in turn, talks to the host, and vice versa. The remote host believes it is communicating with one of its own terminals and allows you to log on.
- **File transfer, access, and management (FTAM).** This application allows a user to access files in a remote host (to make changes or read data), to retrieve files from a remote computer for use in the local computer, and to manage or control files in a remote computer locally.

- **E-mail services.** This application provides the basis for e-mail forwarding and storage.
- **Directory services.** This application provides distributed database sources and access for global information about various objects and services.

### Summary of OSI Layers

Figure 2.6 shows a summary of duties for each layer. In the next section, we describe how some of these duties are mixed and spread into five categories in the TCP/IP protocol suite.

**Figure 2.6** *Summary of OSI layers*




## 2.3 TCP/IP PROTOCOL SUITE

The **TCP/IP protocol suite** was developed prior to the OSI model. Therefore, the layers in the TCP/IP protocol suite do not match exactly with those in the OSI model. The original TCP/IP protocol suite was defined as four software layers built upon the hardware. Today, however, TCP/IP is thought of as a five-layer model with the layers named similarly to the ones in the OSI model. Figure 2.7 shows both configurations.

### Comparison between OSI and TCP/IP Protocol Suite

When we compare the two models, we find that two layers, session and presentation, are missing from the TCP/IP protocol suite. These two layers were not added to the TCP/IP protocol suite after the publication of the OSI model. The application layer in the suite is usually considered to be the combination of three layers in the OSI model, as shown in Figure 2.8.

**Figure 2.7** *Layers in the TCP/IP Protocol Suite*

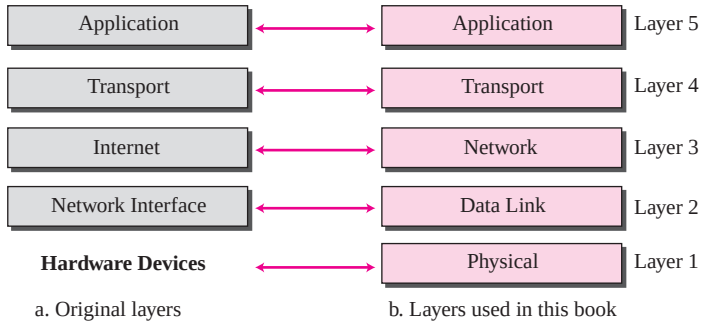
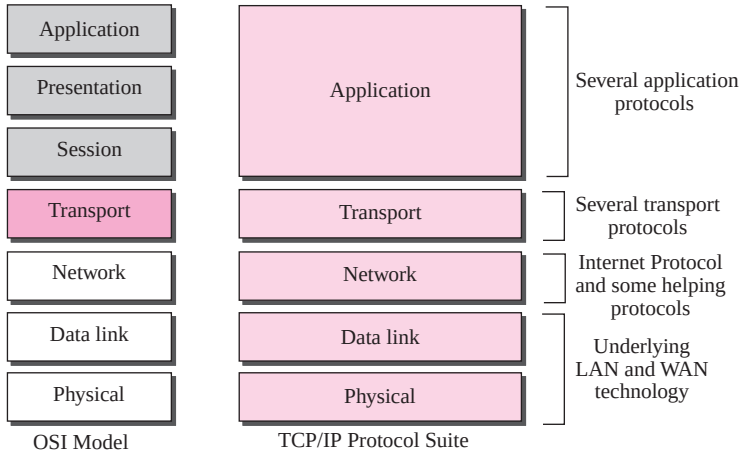



**Figure 2.8** *TCP/IP and OSI model*



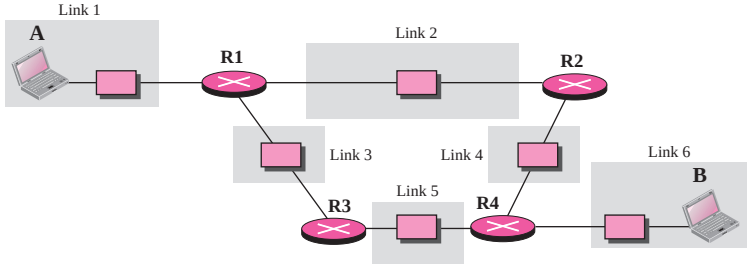

Two reasons were mentioned for this decision. First, TCP/IP has more than one transport-layer protocol. Some of the functionalities of the session layer are available in some of the transport layer protocols. Second, the application layer is not only one piece of software. Many applications can be developed at this layer. If some of the functionalities mentioned in the session and presentation are needed for a particular application, it can be included in the development of that piece of software.

TCP/IP is a hierarchical protocol made up of interactive modules, each of which provides a specific functionality, but the modules are not necessarily interdependent. Whereas the OSI model specifies which functions belong to each of its layers, the layers of the TCP/IP protocol suite contain relatively independent protocols that can be mixed and matched, depending on the needs of the system. The term *hierarchical* means that each upper level protocol is supported by one or more lower level protocols.

## Layers in the TCP/IP Protocol Suite

In this section, we briefly discuss the purpose of each layer in the TCP/IP protocol suite. When we study the purpose of each layer, it is easier to think of a private *internet*, instead of the global Internet. We assume that we want to use the TCP/IP suite in a small, private internet. Such an internet is made up of several small networks, which we call links. A **link** is a network that allows a set of computers to communicate with each other. For example, if all computers in an office are wired together, the connection makes a link. If several computers belonging to a private company are connected via a satellite channel, the connection is a link. A link, as we discussed in Chapter 3, can be a LAN (local area network) serving a small area or a WAN (wide area network) serving a larger area. We also assume that different links are connected together by devices called *routers* or *switches* that route the data to reach their final destinations. Figure 2.9 shows our imaginary internet that is used to show the purpose of each layer. We have six links and four routers (R1 to R4). We have shown only two computers, A and B.

**Figure 2.9** *A private internet*

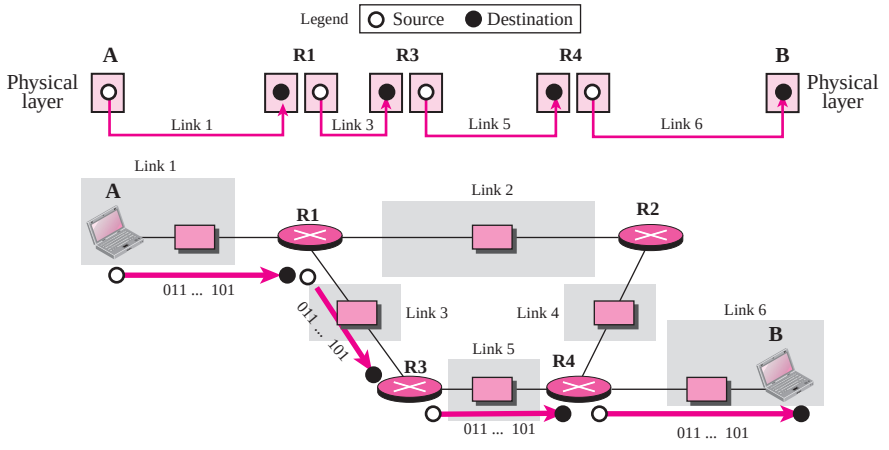



### *Physical Layer*

TCP/IP does not define any specific protocol for the physical layer. It supports all of the standard and proprietary protocols. At this level, the communication is between two hops or nodes, either a computer or router. The unit of communication is a single bit. When the connection is established between the two nodes, a stream of bits is flowing between them. The physical layer, however, treats each bit individually. Figure 2.10 shows the communication between nodes. We are assuming that at this moment the two computers have discovered that the most efficient way to communicate with each other is via routers R1, R3, and R4. How this decision is made is the subject of some future chapters.

Note that if a node is connected to *n* links, it needs *n* physical-layer protocols, one for each link. The reason is that different links may use different physical-layer protocols. The figure, however, shows only physical layers involved in the communication. Each computer involves with only one link; each router involves with only two links. As Figure 2.10 shows, the journey of bits between computer A and computer B is made of four independent short trips. Computer A sends each bit to router R1 in the format of the protocol used by link 1. Router 1 sends each bit to router R3 in the format dictated by the protocol used by link 3. And so on. Router R1 has two three physical layers (two are shown in our scenario). The layer connected to link 1 receives bits according to the format of the protocol

**Figure 2.10** *Communication at the physical layer*




used by link 1; the layer connected to link 3 sends bits according to the format of the protocol used by link 3. It is the same situation with the other two routers involved in the communication.

**The unit of communication at the physical layer is a bit.**

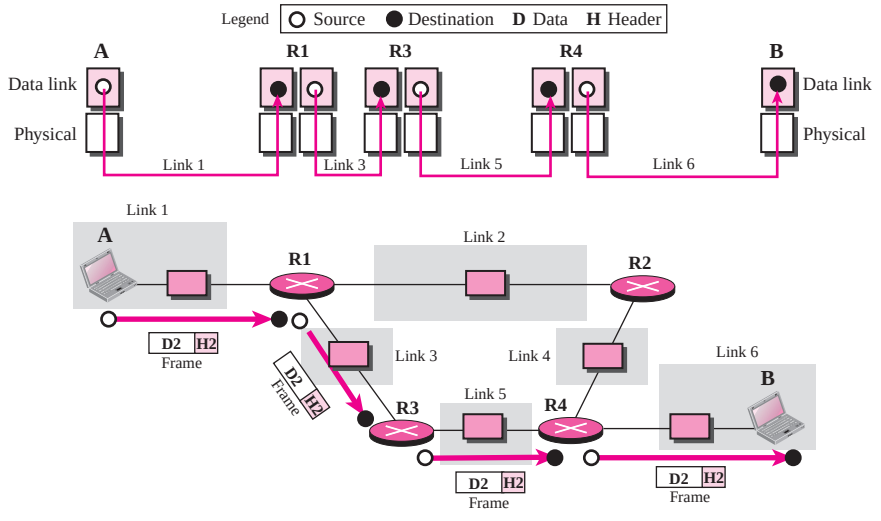
The responsibility of the physical layer, in addition to delivery of bits, matches with what mentioned for the physical layer of the OSI model, but it mostly depends on the underlying technologies that provide links. We see in the next chapter that they are, for example, many protocols for the physical layer of LANs or WANs.

### *Data Link Layer*

TCP/IP does not define any specific protocol for the data link layer either. It supports all of the standard and proprietary protocols. At this level, the communication is also between two hops or nodes. The unit of communication however, is a packet called a **frame**. A frame is a packet that encapsulates the data received from the network layer with an added header and sometimes a trailer. The head, among other communication information, includes the source and destination of frame. The destination address is needed to define the right recipient of the frame because many nodes may have been connected to the link. The source address is needed for possible response or acknowledgment as may be required by some protocols. Figure 2.11 shows the communication at the data link layer.

Note that the frame that is travelling between computer A and router R1 may be different from the one travelling between router R1 and R3. When the frame is received by router R1, this router passes the frame to the data link layer protocol shown at the left. The frame is opened, the data are removed. The data are then passed to the data

**Figure 2.11** *Communication at the data link layer*




link layer protocol shown at the right to create a new frame to be sent to the router R3. The reason is that the two links, link 1 and link 3, may be using different protocols and require frames of different formats. Note also that the figure does not show the physical movement of frames; the physical movement happens only at the physical layer. The two nodes communicate logically at the data link layer, not physically. In other words, the data link layer at router R1 only *thinks* that a frame has been sent directly from the data link layer at computer A. What is sent from A to R1 is a stream of bits from one physical layer to another. Since a frame at A is transformed to a stream of bits, and the bits at R1 are transformed to a frame, it gives this impression to the two data link layer that a frame has been exchanged.

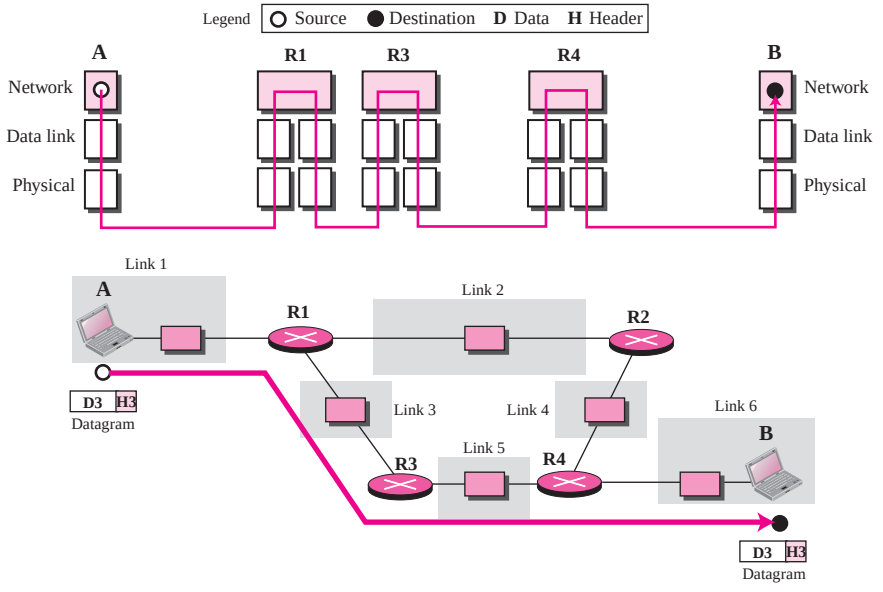
**The unit of communication at the data link layer is a frame.**

### *Network Layer*

At the network layer (or, more accurately, the internetwork layer), TCP/IP supports the Internet Protocol (IP). The **Internet Protocol (IP)** is the transmission mechanism used by the TCP/IP protocols. IP transports data in packets called **datagrams**, each of which is transported separately. Datagrams can travel along different routes and can arrive out of sequence or be duplicated. IP does not keep track of the routes and has no facility for reordering datagrams once they arrive at their destination. Figure 2.12 shows the communication at the network layer.

Note that there is a main difference between the communication at the network layer and the communication at data link or physical layers. Communication at the

**Figure 2.12** *Communication at the network layer*




network layer is end to end while the communication at the other two layers are node to node. The datagram started at computer A is the one that reaches computer B. The network layers of the routers can inspect the source and destination of the packet for finding the best route, but they are not allowed to change the contents of the packet. Of course, the communication is logical, not physical. Although the network layer of computer A and B *think* that they are sending and receiving datagrams, the actual communication again is done at the physical level.

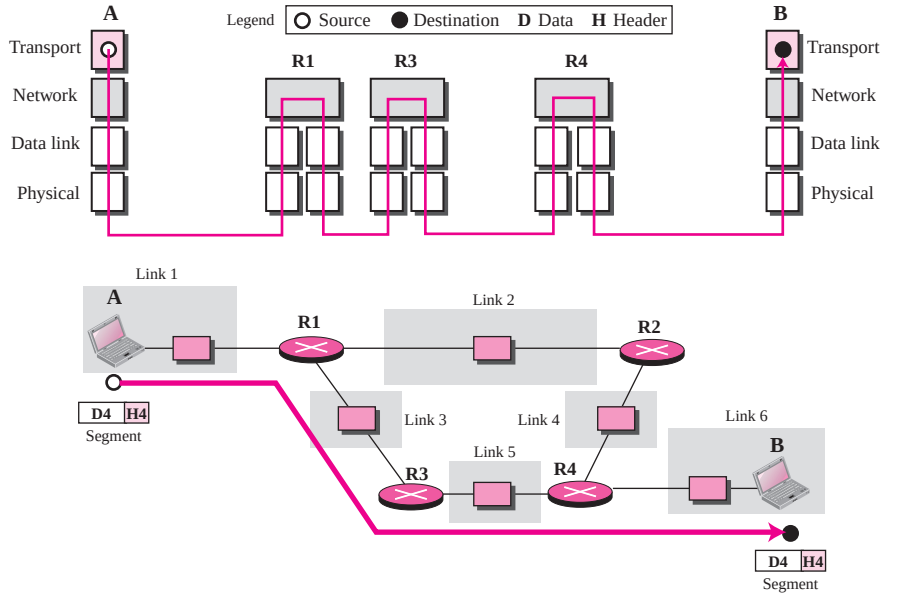
**The unit of communication at the network layer is a datagram.**

### *Transport Layer*

There is a main difference between the transport layer and the network layer. Although all nodes in a network need to have the network layer, only the two end computers need to have the transport layer. The network layer is responsible for sending individual datagrams from computer A to computer B; the transport layer is responsible for delivering the whole message, which is called a segment, a user datagram, or a packet, from A to B. A segment may consist of a few or tens of datagrams. The segments need to be broken into datagrams and each datagram has to be delivered to the network layer for transmission. Since the Internet defines a different route for each datagram, the datagrams may arrive out of order and may be lost. The transport layer at computer B needs to wait until all of these datagrams to arrive, assemble

them and make a segment out of them. Figure 2.13 shows the communication at the transport layer.

**Figure 2.13** *Communication at the transport layer*




Again, we should know that the two transport layers only think that they are communicating with each other using a segment; the communication is done through the physical layer and the exchange of bits.

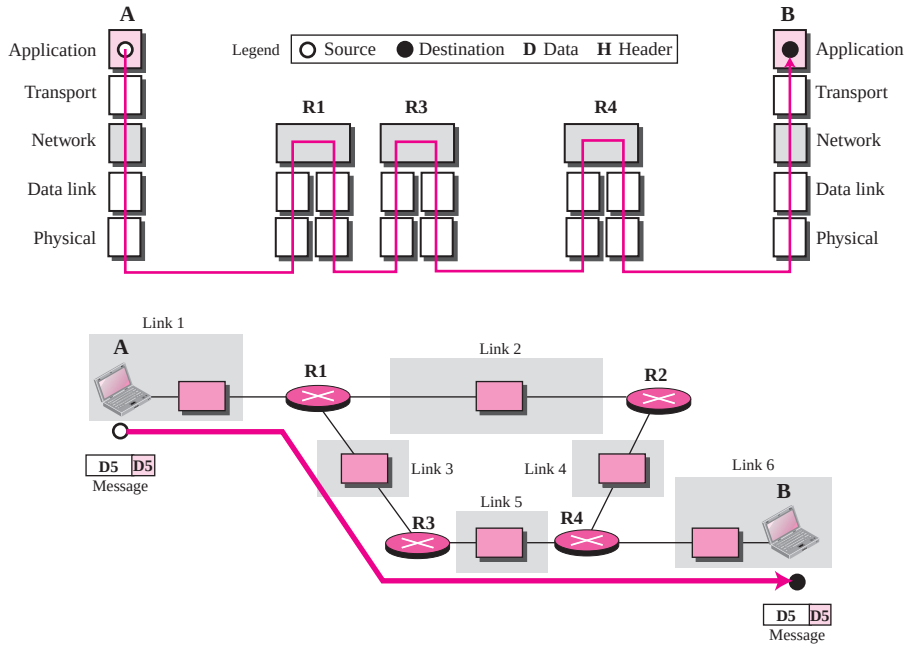
Traditionally, the transport layer was represented in the TCP/IP suite by two protocols: **User Datagram Protocol (UDP)** and **Transmission Control Protocol (TCP)**. A new protocol called **Stream Control Transmission Protocol (SCTP)** has been introduced in the last few years.

**The unit of communication at the transport layer is a segment, user datagram, or a packet, depending on the specific protocol used in this layer.**

### *Application Layer*

The application layer in TCP/IP is equivalent to the combined session, presentation, and application layers in the OSI model. The application layer allows a user to access the services of our private internet or the global Internet. Many protocols are defined at this layer to provide services such as electronic mail, file transfer, accessing the World Wide Web, and so on. We cover most of the standard protocols in later chapters. Figure 2.14 shows the communication at the application layer.

**Figure 2.14** *Communication at the application layer*




Note that the communication at the application layer, like the one at the transport layer, is end to end. A message generated at computer A is sent to computer B without being changed during the transmission.

**The unit of communication at the application layer is a message.**

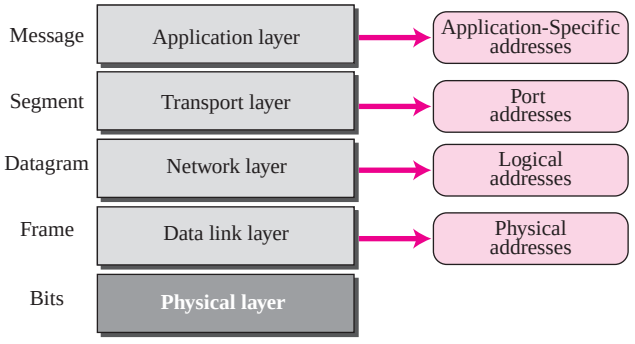
## 2.4 ADDRESSING

Four levels of addresses are used in an internet employing the TCP/IP protocols: **physical address, logical address, port address,** and **application-specific address.** Each address is related to a one layer in the TCP/IP architecture, as shown in Figure 2.15.

### Physical Addresses

The physical address, also known as the link address, is the address of a node as defined by its LAN or WAN. It is included in the frame used by the data link layer. It is the lowest-level address. The physical addresses have authority over the link (LAN or WAN). The size and format of these addresses vary depending on the network. For example, Ethernet uses a 6-byte (48-bit) physical address that is imprinted on the network interface card (NIC). LocalTalk (Apple), however, has a 1-byte dynamic address that changes each time the station comes up.

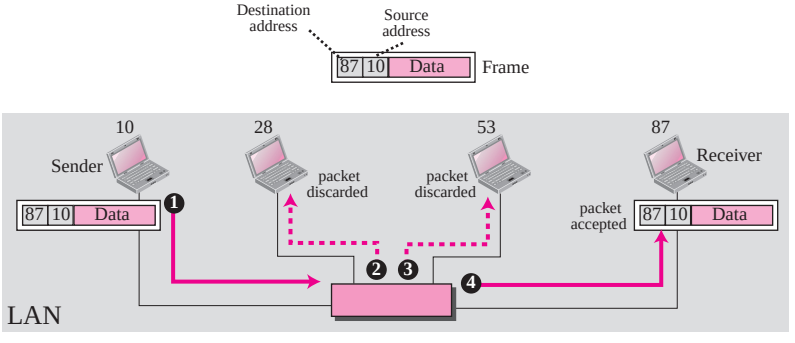
**Figure 2.15** *Addresses in the TCP/IP Protocol Suite*




## Example 2.3

In Figure 2.16 a node with physical address 10 sends a frame to a node with physical address 87. The two nodes are connected by a link (a LAN). At the data link layer, this frame contains physical (link) addresses in the header. These are the only addresses needed. The rest of the header contains other information needed at this level. The trailer usually contains extra bits needed for error detection. As the figure shows, the computer with physical address 10 is the sender, and the computer with physical address 87 is the receiver. The data link layer at the sender receives data from an upper layer. It encapsulates the data in a frame, adding a header and a trailer. The header, among other pieces of information, carries the receiver and the sender physical (link) addresses. Note that in most data link protocols, the destination address 87 in this case, comes before the source address (10 in this case). The frame is propagated through the LAN. Each station with a physical address other than 87 drops the frame because the destination address in the frame does not match its own physical address. The intended destination computer, however, finds a match between the destination address in the frame and its own physical address. The frame is checked, the header and trailer are dropped, and the data part is decapsulated and delivered to the upper layer.

**Figure 2.16** *Example 2.3: physical addresses*



### Example 2.4

As we will see in Chapter 3, most local area networks use a 48-bit (6-byte) physical address written as 12 hexadecimal digits; every byte (2 hexadecimal digits) is separated by a colon, as shown below:

**07:01:02:01:2C:4B**
A 6-byte (12 hexadecimal digits) physical address

#### *Unicast, Multicast, and Broadcast Physical Addresses*

Physical addresses can be either **unicast** (one single recipient), **multicast** (a group of recipients), or **broadcast** (to be received by all systems in the network). Some networks support all three addresses. For example, Ethernet (see Chapter 3) supports the unicast physical addresses (6 bytes), the multicast addresses, and the broadcast addresses. Some networks do not support the multicast or broadcast physical addresses. If a frame must be sent to a group of recipients or to all systems, the multicast or broadcast address must be simulated using unicast addresses. This means that multiple packets are sent out using unicast addresses.

## Logical Addresses

Logical addresses are necessary for universal communications that are independent of underlying physical networks. Physical addresses are not adequate in an internetwork environment where different networks can have different address formats. A universal addressing system is needed in which each host can be identified uniquely, regardless of the underlying physical network. The logical addresses are designed for this purpose. A logical address in the Internet is currently a 32-bit address that can uniquely define a host connected to the Internet. No two publicly addressed and visible hosts on the Internet can have the same IP address.

### Example 2.5

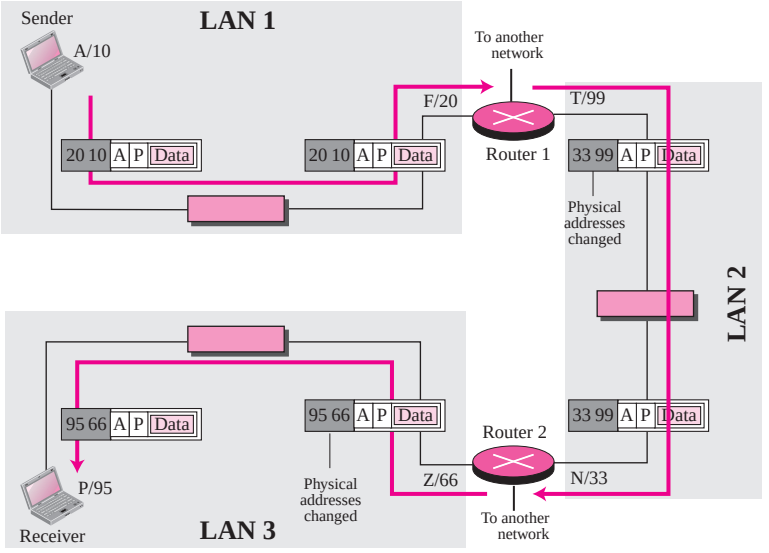
Figure 2.17 shows a part of an internet with two routers connecting three LANs.

Each device (computer or router) has a pair of addresses (logical and physical) for each connection. In this case, each computer is connected to only one link and therefore has only one pair of addresses. Each router, however, is connected to three networks (only two are shown in the figure). So each router has three pairs of addresses, one for each connection. Although it may be obvious that each router must have a separate physical address for each connection, it may not be obvious why it needs a logical address for each connection. We discuss these issues in Chapters 11 and 12 when we discuss routing.

The computer with logical address A and physical address 10 needs to send a packet to the computer with logical address P and physical address 95. We use letters to show the logical addresses and numbers for physical addresses, but note that both are actually numbers, as we will see in later chapters.

The sender encapsulates its data in a packet at the network layer and adds two logical addresses (A and P). Note that in most protocols, the logical source address comes before the logical destination address (contrary to the order of physical addresses). The network layer, however, needs to find the physical address of the next hop before the packet can be delivered. The network layer consults its routing table (see Chapter 6) and finds the logical address of the next hop

**Figure 2.17** *Example 2.5: logical addresses*




(router 1) to be F. Another protocol, **Address Resolution Protocol** (ARP), which will be discussed in later chapters, finds the physical address of router 1 that corresponds to its logical address (20). Now the network layer passes this address to the data link layer, which in turn, encapsulates the packet with physical destination address 20 and physical source address 10.

The frame is received by every device on LAN 1, but is discarded by all except router 1, which finds that the destination physical address in the frame matches with its own physical address. The router decapsulates the packet from the frame to read the logical destination address P. Since the logical destination address does not match the router's logical address, the router knows that the packet needs to be forwarded. The router consults its routing table and ARP to find the physical destination address of the next hop (router 2), creates a new frame, encapsulates the packet, and sends it to router 2.

Note the physical addresses in the frame. The source physical address changes from 10 to 99. The destination physical address changes from 20 (router 1 physical address) to 33 (router 2 physical address). The logical source and destination addresses must remain the same; otherwise the packet will be lost.

At router 2 we have a similar scenario. The physical addresses are changed, and a new frame is sent to the destination computer. When the frame reaches the destination, the packet is decapsulated. The destination logical address P matches the logical address of the computer. The data are decapsulated from the packet and delivered to the upper layer. Note that although physical addresses will change from hop to hop, logical addresses remain the same from the source to destination. There are some exceptions to this rule that we discover later in the book.

**The physical addresses will change from hop to hop, but the logical addresses remain the same.**

### *Unicast, Multicast, and Broadcast Addresses*

The logical addresses can be either unicast (one single recipient), multicast (a group of recipients), or broadcast (all systems in the network). There are limitations on broadcast addresses.

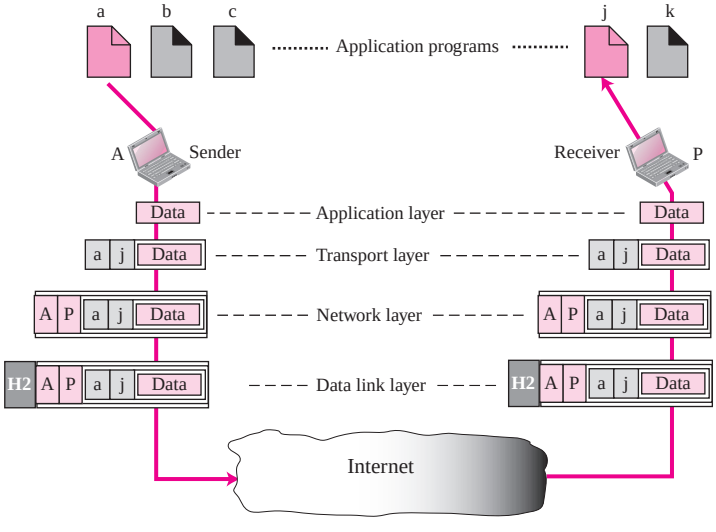
## Port Addresses

The IP address and the physical address are necessary for a quantity of data to travel from a source to the destination host. However, arrival at the destination host is not the final objective of data communications on the Internet. A system that sends nothing but data from one computer to another is not complete. Today, computers are devices that can run multiple processes at the same time. The end objective of Internet communication is a process communicating with another process. For example, computer A can communicate with computer C by using TELNET. At the same time, computer A communicates with computer B by using the File Transfer Protocol (FTP). For these processes to receive data simultaneously, we need a method to label the different processes. In other words, they need addresses. In the TCP/IP architecture, the label assigned to a process is called a port address. A port address in TCP/IP is 16 bits in length.

### Example 2.6

Figure 2.18 shows two computers communicating via the Internet. The sending computer is running three processes at this time with port addresses a, b, and c. The receiving computer is running two processes at this time with port addresses j and k. Process a in the sending computer needs to communicate with process j in the receiving computer. Note that although both computers are using the same application, FTP, for example, the port addresses are different because one is a client

**Figure 2.18** *Example 2.6: port numbers*



program and the other is a server program, as we will see in Chapter 17. To show that data from process a need to be delivered to process j, and not k, the transport layer encapsulates data from the application layer in a packet and adds two port addresses (a and j), source and destination. The packet from the transport layer is then encapsulated in another packet at the network layer with logical source and destination addresses (A and P). Finally, this packet is encapsulated in a frame with the physical source and destination addresses of the next hop. We have not shown the physical addresses because they change from hop to hop inside the cloud designated as the Internet. Note that although physical addresses change from hop to hop, logical and port addresses remain the same from the source to destination. There are some exceptions to this rule that we discuss later in the book.

**The physical addresses change from hop to hop, but the logical and port addresses usually remain the same.**

### Example 2.7

As we will see in future chapters, a port address is a 16-bit address represented by one decimal number as shown.

**753**
A 16-bit port address represented as one single number

### Application-Specific Addresses

Some applications have user-friendly addresses that are designed for that specific application. Examples include the e-mail address (for example, forouzan@fhda.edu) and the Universal Resource Locator (URL) (for example, www.mhhe.com). The first defines the recipient of an e-mail; the second is used to find a document on the World Wide Web. These addresses, however, get changed to the corresponding port and logical addresses by the sending computer, as we will see in later chapters.

## 2.5 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], [Pet & Dav 03], [Kur & Ros 08], and [Gar & Vid 04].

### RFCs

Two RFCs in particular discuss the TCP/IP suite: RFC 791 (IP) and RFC 817 (TCP). In future chapters we list different RFCs related to each protocol in each layer.

## 2.6 KEY TERMS

access control
Address Resolution Protocol (ARP)
application layer
application-specific address
bit
broadcast physical address
bus topology
compression
connection control
datagram
data link layer
dialog control
directory services
encoding
encryption
error control
file transfer, access, and management (FTAM)
flow control
frame
full-duplex mode
half-duplex mode
interface
International Standards Organization (ISO)
internetwork
line configuration
link
logical address
logical addressing
mesh topology
multicast physical address
multipoint configuration
network layer
network virtual terminal
open system
Open Systems Interconnection (OSI) model
peer-to-peer processes
physical address
physical layer
physical topology
point-to-point configuration
port address
presentation layer
process-to-process delivery
ring topology
routing
segmentation
service-point addressing
session layer
simplex mode
source-to-destination delivery
star topology
Stream Control Transmission Protocol (SCTP)
synchronization points
TCP/IP protocol suite
translation
Transmission Control Protocol (TCP)
transmission mode
transmission rate
transport layer
unicast physical address
User Datagram Protocol (UDP)

## 2.7 SUMMARY

- The International Standards Organization (ISO) created a model called the Open Systems Interconnection (OSI), which allows diverse systems to communicate. The seven-layer OSI model provides guidelines for the development of universally compatible networking protocols. The physical, data link, and network layers are the network support layers.The session, presentation, and application layers are the user support layers.The transport layer links the network support layers and the user support layers.
- The physical layer coordinates the functions required to transmit a bit stream over a physical medium. The data link layer is responsible for delivering data units

from one station to the next without errors. The network layer is responsible for the source-to-destination delivery of a packet across multiple network links. The transport layer is responsible for the process-to-process delivery of the entire message. The session layer establishes, maintains, and synchronizes the interactions between communicating devices. The presentation layer ensures interoperability between communicating devices through transformation of data into a mutually agreed-upon format. The application layer enables the users to access the network.

- TCP/IP is a five-layer hierarchical protocol suite developed before the OSI model. The TCP/IP application layer is equivalent to the combined session, presentation, and application layers of the OSI model.
- Four types of addresses are used by systems using the TCP/IP protocol: the physical address, the internetwork address (IP address), the port address, and application-specific address. The physical address, also known as the link address, is the address of a node as defined by its LAN or WAN. The IP address uniquely defines a host on the Internet. The port address identifies a process on a host. The application-specific address is used by some applications to provide user-friendly access.

## 2.8 PRACTICE SET

### Exercises

1. How are OSI and ISO related to each other?
2. Match the following to one or more layers of the OSI model:
   a. route determination
   b. flow control
   c. interface to transmission media
   d. provides access for the end user
3. Match the following to one or more layers of the OSI model:
   a. reliable process-to-process message delivery
   b. route selection
   c. defines frames
   d. provides user services such as e-mail and file transfer
   e. transmission of bit stream across physical medium
4. Match the following to one or more layers of the OSI model:
   a. communicates directly with user's application program
   b. error correction and retransmission
   c. mechanical, electrical, and functional interface
   d. responsibility for carrying frames between adjacent nodes
5. Match the following to one or more layers of the OSI model:
   a. format and code conversion services
   b. establishes, manages, and terminates sessions

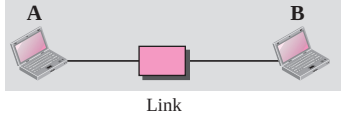
**c.** ensures reliable transmission of data

**d.** log-in and log-out procedures

**e.** provides independence from differences in data representation

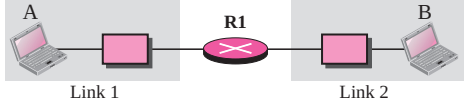
**6.** Show the communication at the application layer (see Figure 2.14) for the simple private internet in Figure 2.19.

**Figure 2.19** *Exercise 6*




**7.** Show the communication at the application layer (see Figure 2.14) for the simple private internet in Figure 2.20.

**Figure 2.20** *Exercise 7*




**8.** A 100-byte message is sent through a private internet using the TCP/IP protocol suite. If the protocol adds a 10-byte header at each layer, what is the efficiency of the system (the ratio of the number of useful bytes to the number of total bytes)?

**9.** If a port number is 16 bits (2 bytes), what is the minimum header size at transport layer of the TCP/IP protocol suite?

**10.** If a logical address is 32 bits (4 bytes), what is the minimum header size at network layer of the TCP/IP protocol suite?

**11.** If a physical address is 48 bits (6 bytes) what is the minimum header size at the data link layer of the TCP/IP protocol suite?

**12.** Do we encapsulate our message when we send a regular letter to a friend? When we send a post card to a friend while we are vacationing in another country, do we encapsulate our message?

**13.** Why do you think that we do not need addresses at the physical layer?

**14.** Why do you think a radio station does not need the addresses of its listeners when a message is broadcast?

**15.** Why do you think both the sender and receiver addresses are needed in the Internet?

**16.** Why do you think there is a need for four levels of addresses in the Internet, but only one level of addresses (telephone numbers) in a telephone network?

## Research Activities

**17.** Domain Name System or DNS (see Chapter 19) is an application program in the TCP/IP protocol suite. Research and find the equivalent of this protocol (if any) in the OSI model. Compare and contrast the two.

**18.** File Transfer Protocol or FTP (see Chapter 21) is an application program in the TCP/IP protocol suite. Research and find the equivalent of this protocol (if any) in the OSI model. Compare and contrast the two.

**19.** Trivial File Transfer Protocol or TFTP (see Chapter 21) is an application program in the TCP/IP protocol suite. Research and find the equivalent of this protocol (if any) in the OSI model. Compare and contrast the two.

**20.** There are several transport layer models proposed in the OSI model. Research and find all of them. Explain the differences between them.

**21.** There are several network layer models proposed in the OSI model. Research and find all of them. Explain the differences between them.

CHAPTER

# 3 Underlying Technologies

We can think of the Internet as a series of backbone networks that are run by international, national, and regional ISPs. The backbones are joined together by connecting devices such as routers or switching stations. The end users are either part of the local ISP LAN or connected via point-to-point networks to the LANs. Conceptually, the Internet is a set of switched WANs (backbones), LANs, point-to-point WANs, and connecting or switching devices.

Although the TCP/IP Protocol Suite is normally shown as a five-layer stack, it only defines the three upper layers; TCP/IP is only concerned with the network, transport, and application layers. This means that TCP/IP assumes the existence of these WANs, LANs, and the connecting devices that join them.

As a brief review, we touch upon some of these underlying technologies in this chapter.

## OBJECTIVES

*The chapter has several objectives:*

- To briefly discuss the technology of dominant wired LANs, Ethernet, including traditional, fast, gigabit, and ten-gigabit Ethernet.
- To briefly discuss the technology of wireless WANs, including IEEE 802.11 LANs, and Bluetooth.
- To briefly discuss the technology of point-to-point WANs including 56K modems, DSL, cable modem, T-lines, and SONET.
- To briefly discuss the technology of switched WANs including X.25, Frame Relay, and ATM.
- To discuss the need and use of connecting devices such as repeaters (hubs), bridges (two-layer switches), and routers (three-layer switches).

## 3.1 WIRED LOCAL AREA NETWORKS

A local area network (LAN) is a computer network that is designed for a limited geographic area such as a building or a campus. Although a LAN can be used as an isolated network to connect computers in an organization for the sole purpose of sharing resources, most LANs today are also linked to a wide area network (WAN) or the Internet.

The LAN market has seen several technologies such as Ethernet, token ring, token bus, FDDI, and ATM LAN. Some of these technologies survived for a while, but Ethernet is by far the dominant technology.

In this section, we first briefly discuss the IEEE Standard Project 802, designed to regulate the manufacturing and interconnectivity between different LANs. We then concentrate on the Ethernet LANs.

Although Ethernet has gone through a four-generation evolution during the last few decades, the main concept has remained the same. Ethernet has changed to meet the market needs and to make use of the new technologies.

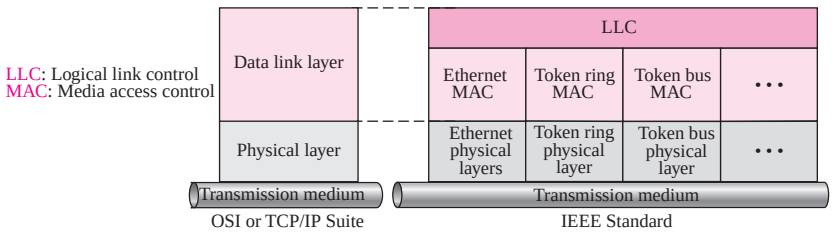
### IEEE Standards

In 1985, the Computer Society of the IEEE started a project, called **Project 802,** to set standards to enable intercommunication among equipment from a variety of manufacturers. Project 802 does not seek to replace any part of the OSI or the Internet model. Instead, it is a way of specifying functions of the physical layer and the data link layer of major LAN protocols.

The standard was adopted by the American National Standards Institute (ANSI). In 1987, the International Standards Organization (ISO) also approved it as an international standard under the designation ISO 8802.

The relationship of the 802 Standard to the traditional OSI model is shown in Figure 3.1. The IEEE has subdivided the data link layer into two sublayers: **logical link control (LLC)** and **media access control** (**MAC**)**.** IEEE has also created several physical layer standards for different LAN protocols.

**Figure 3.1** *IEEE standard for LANs*



In this text, however, we treat physical and data link layer together as the underlying technology supporting other layers in the TCP/IP protocol suite. For more details about physical and data link layer technology see Forouzan, *Data Communications and Networking,* 4th ed., McGraw-Hill, 2007.

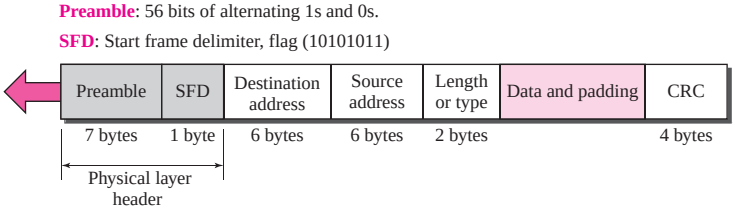
## Frame Format

The packet sent in an Ethernet LAN is called a frame. In this section we discuss the format and the length of the frame that is used in our versions of the Ethernet.

### *Frame Format*

The Ethernet frame contains seven fields: preamble, SFD, DA, SA, length or type of data unit, upper-layer data, and the CRC. Ethernet does not provide any mechanism for acknowledging received frames, making it what is known as an unreliable medium. Acknowledgments must be implemented at the higher layers. The format of the MAC frame is shown in Figure 3.2.

**Figure 3.2** *Ethernet frame*



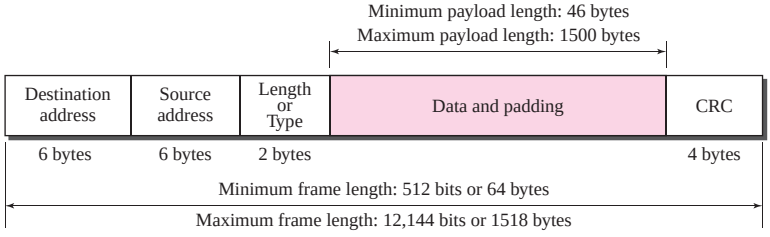

- **Preamble.** The first field of the 802.3 frame contains 7 bytes (56 bits) of alternating 0s and 1s that alerts the receiving system to the coming frame and enables it to synchronize its input timing. The pattern provides only an alert and a timing pulse. The 56-bit pattern allows the stations to miss some bits at the beginning of the frame. The preamble is actually added at the physical layer and is not (formally) part of the frame.
- **Start frame delimiter (SFD).** The second field (1 byte: 10101011) signals the beginning of the frame. The SFD warns the station or stations that this is the last chance for synchronization. The last 2 bits are 11 and alert the receiver that the next field is the destination address. The SFD is also added at the physical layer.
- **Destination address (DA).** The DA field is 6 bytes and contains the physical address of the destination station or stations to receive the packet. We will discuss addressing shortly.
- **Source address (SA).** The SA field is also 6 bytes and contains the physical address of the sender of the packet.
- **Length or type.** This field is defined as a type field or length field. The original Ethernet used this field as the type field to define the upper-layer protocol using the MAC frame. The IEEE standard used it as the length field to define the number of bytes in the data field. Both uses are common today.

- **Data.** This field carries data encapsulated from the upper-layer protocols. It is a minimum of 46 and a maximum of 1500 bytes, as we will see later.
- **CRC.** The last field contains error detection information, in this case a CRC-32 (See Appendix C).

### *Frame Length*

Ethernet has imposed restrictions on both the minimum and maximum lengths of a frame, as shown in Figure 3.3.

**Figure 3.3** *Minimum and maximum lengths*




The minimum length restriction is required for the correct operation of CSMA/CD, as we will see shortly. An Ethernet frame needs to have a minimum length of 512 bits or 64 bytes. Part of this length is the header and the trailer. If we count 18 bytes of header and trailer (6 bytes of source address, 6 bytes of destination address, 2 bytes of length or type, and 4 bytes of CRC), then the minimum length of data from the upper layer is 64 − 18 = 46 bytes. If the upper-layer packet is less than 46 bytes, padding is added to make up the difference.

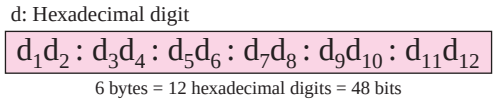
The standard defines the maximum length of a frame (without preamble and SFD field) as 1518 bytes. If we subtract the 18 bytes of header and trailer, the maximum length of the payload is 1500 bytes. The maximum length restriction has two historical reasons. First, memory was very expensive when Ethernet was designed: a maximum length restriction helped to reduce the size of the buffer. Second, the maximum length restriction prevents one station from monopolizing the shared medium, blocking other stations that have data to send.

**Minimum length: 64 bytes (512 bits)** **Maximum length: 1518 bytes (12,144 bits)**

## Addressing

Each station on an Ethernet network (such as a PC, workstation, or printer) has its own **network interface card (NIC).** The NIC fits inside the station and provides the station with a 6-byte physical address. As shown in Figure 3.4, the Ethernet address is 6 bytes (48 bits), normally written in **hexadecimal notation,** with a colon between the bytes. The address normally is referred to as the data link address, physical address, or MAC address.

**Figure 3.4** *Ethernet address in hexadecimal notation*



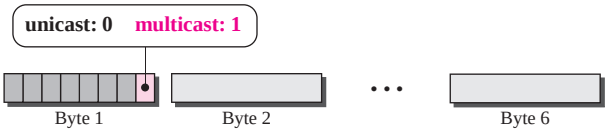

For example, the following shows an Ethernet MAC address:

**4A:30:10:21:10:1A**

### *Unicast, Multicast, and Broadcast Addresses*

A source address is always a unicast address—the frame comes from only one station. The destination address, however, can be unicast, multicast, or broadcast. Figure 3.5 shows how to distinguish a unicast address from a multicast address. If the least significant bit of the first byte in a destination address is 0, the address is unicast; otherwise, it is multicast.

**Figure 3.5** *Unicast and multicast addresses*




**The least significant bit of the first byte defines the type of address. If the bit is 0, the address is unicast; otherwise, it is multicast.**

A unicast destination address defines only one recipient; the relationship between the sender and the receiver is one-to-one. A multicast destination address defines a group of addresses; the relationship between the sender and the receivers is one-to-many.

The broadcast address is a special case of the multicast address; the recipients are all the stations on the LAN. A broadcast destination address is forty-eight 1s.

**The broadcast destination address is a special case of the multicast address in which all bits are 1s.**

### Example 3.1

Define the type of the following destination addresses:

**a.** 4A:30:10:21:10:1A
**b.** 47:20:1B:2E:08:EE
**c.** FF:FF:FF:FF:FF:FF

**Solution**

To find the type of the address, we need to look at the second hexadecimal digit from the left. If it is even, the address is unicast. If it is odd, the address is multicast. If all digits are F's, the address is broadcast. Therefore, we have the following:

- **a.** This is a unicast address because A in binary is 1010 (even).
- **b.** This is a multicast address because 7 in binary is 0111 (odd).
- **c.** This is a broadcast address because all digits are F's.

The way the addresses are sent out on line is different from the way they are written in hexadecimal notation. The transmission is left-to-right, byte by byte; however, for each byte, the least significant bit is sent first and the most significant bit is sent last. This means that the bit that defines an address as unicast or multicast arrives first at the receiver.

### Example 3.2

Show how the address 47:20:1B:2E:08:EE is sent out on line.

**Solution**

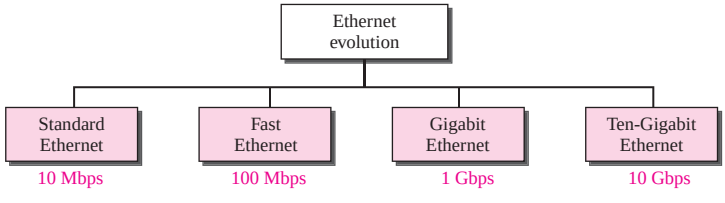
The address is sent left-to-right, byte by byte; for each byte, it is sent right-to-left, bit by bit, as shown below:

```
← 11100010 00000100 11011000 01110100 00010000 01110111
```

## Ethernet Evolution

Ethernet was created in 1976 at Xerox's Palo Alto Research Center (PARC). Since then, it has gone through four generations: **Standard Ethernet** (10 Mbps), **Fast Ethernet** (100 Mbps), **Gigabit Ethernet** (1 Gbps), and **Ten-Gigabit Ethernet** (10 Gbps), as shown in Figure 3.6. We briefly discuss all these generations starting with the first, Standard (or traditional) Ethernet.

**Figure 3.6** *Ethernet evolution through four generations*




## Standard Ethernet

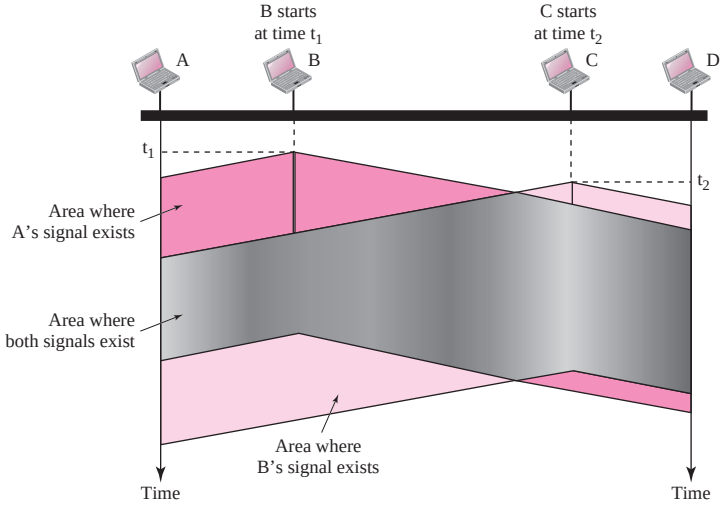
The original Ethernet with 10-Mbps data rate is now history, but we briefly discuss its characteristics to pave the way for understanding other Ethernet versions.

### *Access Method: CSMA/CD*

The IEEE 802.3 standard defines **carrier sense multiple access with collision detection (CSMA/CD)** as the access method for traditional Ethernet. Stations on a traditional Ethernet can be connected together using a physical bus or star topology, but the logical topology is always a bus. By this, we mean that the medium (channel) is shared between stations and only one station at a time can use it. It also implies that all stations receive a frame sent by a station (broadcasting). The real destination keeps the frame while the rest drop it. In this situation, how can we be sure that two stations are not using the medium at the same time? If they do, their frames will collide with each other.

To minimize the chance of collision and, therefore, increase the performance, the CSMA method was developed. The chance of collision can be reduced if a station senses the medium before trying to use it. **Carrier sense multiple access (CSMA)** requires that each station first listen to the medium (or check the state of the medium) before sending. In other words, CSMA is based on the principle "sense before transmit" or "listen before talk." CSMA can reduce the possibility of collision, but it cannot eliminate it. The reason for this is shown in Figure 3.7, a space and time model of a CSMA network. Stations are connected to a shared channel (usually a dedicated medium).

**Figure 3.7** *Space/time model of a collision in CSMA*

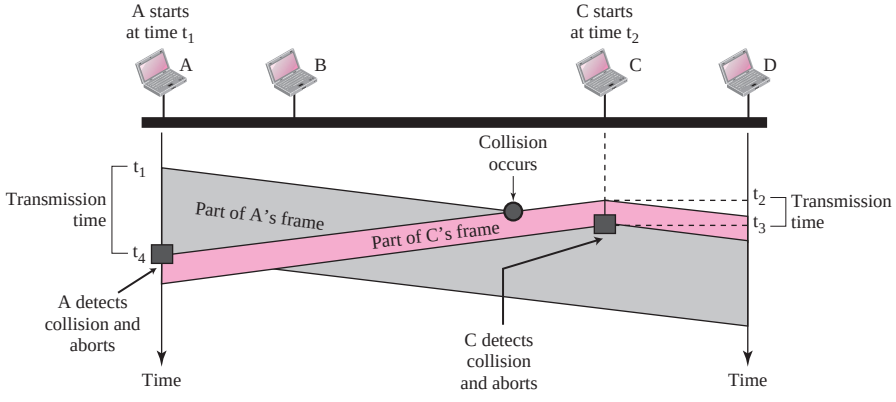



The possibility of collision still exists because of propagation delay; when a station sends a frame, it still takes time (although very short) for the first bit to reach every station and for every station to sense it. In other words, a station may sense the medium and find it idle, only because the first bit sent by another station has not yet been received.

At time $t_1$, station B senses the medium and finds it idle, so it sends a frame. At time $t_2$ ($t_2 > t_1$), station C senses the medium and finds it idle because, at this time, the first bits from station B have not reached station C. Station C also sends a frame. The two signals collide and both frames are destroyed.

Carrier sense multiple access with collision detection (CSMA/CD) augments the algorithm to handle the collision. In this method, a station monitors the medium after it sends a frame to see if the transmission was successful. If so, the station is finished. If, however, there is a collision, the frame is sent again. To better understand CSMA/CD, let us look at the first bits transmitted by the two stations involved in the collision. Although each station continues to send bits in the frame until it detects the collision, we show what happens as the first bits collide. In Figure 3.8, stations A and C are involved in the collision.

**Figure 3.8** *Collision of the first bit in CSMA/CD*




At time $t_1$, station A has started sending the bits of its frame. At time $t_2$, station C has not yet sensed the first bit sent by A. Station C starts sending the bits in its frame, which propagate both to the left and to the right. The collision occurs sometime after time $t_2$. Station C detects a collision at time $t_3$ when it receives the first bit of A's frame. Station C immediately (or after a short time, but we assume immediately) aborts transmission. Station A detects collision at time $t_4$ when it receives the first bit of C's frame; it also immediately aborts transmission. Looking at the figure, we see that A transmits for the duration $t_4 - t_1$; C transmits for the duration $t_3 - t_2$. Later we show that, for the protocol to work, the length of any frame divided by the bit rate in this protocol must be more than either of these durations. At time $t_4$, the transmission of A's frame, though incomplete, is aborted; at time $t_3$, the transmission of B's frame, though incomplete, is aborted.

### *Minimum Frame Size*

For CSMA/CD to work, we need a restriction on the frame size. Before sending the last bit of the frame, the sending station must detect a collision, if any, and abort the transmission. This is so because the station, once the entire frame is sent, does not keep a copy of the frame and does not monitor the line for collision detection. Therefore, the frame

transmission time $T_{fr}$ must be at least two times the maximum propagation time $T_p$. To understand the reason, let us think about the worst-case scenario. If the two stations involved in a collision are the maximum distance apart, the signal from the first takes time $T_p$ to reach the second, and the effect of the collision takes another time $T_P$ to reach the first. So the requirement is that the first station must still be transmitting after $2T_p$.

### Example 3.3

In the standard Ethernet, if the maximum propagation time is 25.6 μs, what is the minimum size of the frame?

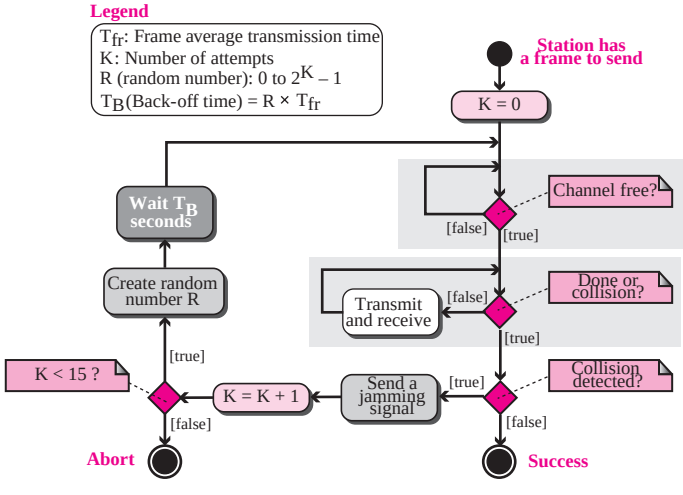
**Solution**

The frame transmission time is $T_{fr} = 2 \times T_p = 51.2$ μs. This means, in the worst case, a station needs to transmit for a period of 51.2 μs to detect the collision. The minimum size of the frame is 10 Mbps × 51.2 μs = 512 bits or 64 bytes. This is actually the minimum size of the frame for Standard Ethernet, as we discussed before.

### *Procedure*

Figure 3.9 shows the flow diagram for CSMA/CD. We need to sense the channel before we start sending the frame. We do not send the entire frame and then look for a collision. The station transmits and receives continuously and simultaneously (using two different ports). We use a loop to show that transmission is a continuous process. We constantly monitor in order to detect one of two conditions: either transmission is finished or a collision is detected. Either event stops transmission. When we come out of the loop, if a collision has not been detected, it means that transmission is complete; the entire frame is transmitted. Otherwise, a collision has occurred. The diagram also shows a short **jamming signal** that enforces the collision in case other stations have not yet sensed the collision.

**Figure 3.9** *CSMA/CD flow diagram*



### *Implementation*

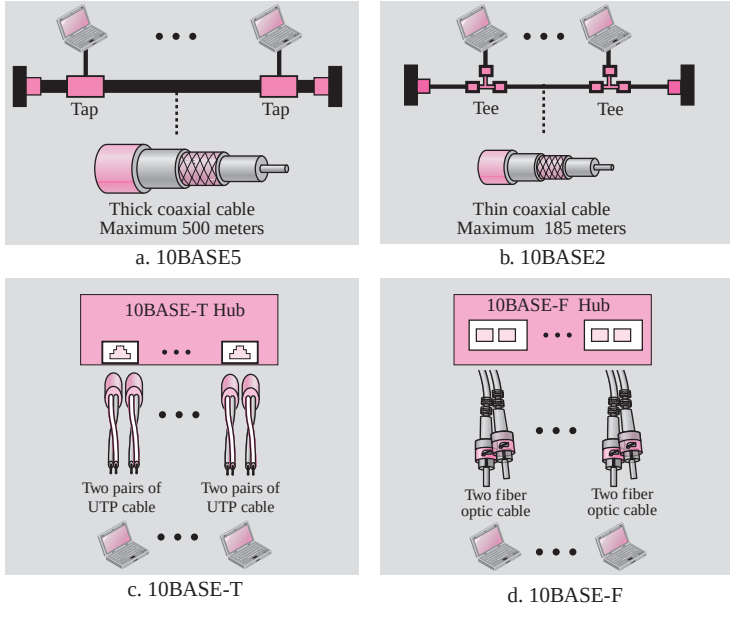
The Standard Ethernet defined several implementations, but only four of them became popular during ’80s. Table 3.1 shows a summary of Standard Ethernet implementations. In the nomenclature 10Base-X, the number defines the data rate (10 Mbps), the term Base means baseband (digital) signal, and X approximately defines either the maximum size of the cable in 100 meters (for example 5 for 500 or 2 for 185 meters) or the type of the cable, T for unshielded twisted pair cable (UTP) and F for fiber-optic.

**Table 3.1** *Summary of Standard Ethernet implementations*

| *Characteristics* | *10Base5* | *10Base2* | *10Base-T* | *10Base-F* |
|---|---|---|---|---|
| Medium | Thick coax | Thin coax | 2 UTP | 2 Fiber |
| Maximum length | 500 m | 185 m | 100 m | 2000 m |

Figure 3.10 shows simplified diagrams of each implementation.

**Figure 3.10** *Standard Ethernet Implementation*




## Fast Ethernet

Fast Ethernet was designed to compete with LAN protocols such as FDDI or Fiber Channel. IEEE created Fast Ethernet under the name 802.3u. Fast Ethernet is backward-compatible with Standard Ethernet, but it can transmit data 10 times faster at a rate of 100 Mbps. The goals of Fast Ethernet can be summarized as follows:

1. Upgrade the data rate to 100 Mbps.
2. Make it compatible with Standard Ethernet.

3. Keep the same 48-bit address.
4. Keep the same frame format.
5. Keep the same minimum and maximum frame lengths.

### *MAC Sublayer*

A main consideration in the evolution of Ethernet from 10 to 100 Mbps was to keep the MAC sublayer untouched. However, a decision was made to drop the bus topologies and keep only the star topology. For the star topology, there are two choices: half duplex and full duplex. In the half-duplex approach, the stations are connected via a hub; in the full-duplex approach, the connection is made via a switch with buffers at each port (see Section 3.5, Connecting Devices, at the end of the chapter).

The access method is the same (CSMA/CD) for the half-duplex approach; for full-duplex Fast Ethernet, there is no need for CSMA/CD. However, the implementations keep CSMA/CD for backward compatibility with Standard Ethernet.

### *Autonegotiation*

A new feature added to Fast Ethernet is called **autonegotiation.** It allows a station or a hub a range of capabilities. Autonegotiation allows two devices to negotiate the mode or data rate of operation. It was designed particularly for the following purposes:

- To allow incompatible devices to connect to one another. For example, a device with a maximum capacity of 10 Mbps can communicate with a device with a 100 Mbps capacity (but can work at a lower rate).
- To allow one device to have multiple capabilities.
- To allow a station to check a hub's capabilities.

### *Implementation*

Fast Ethernet implementation at the physical layer can be categorized as either two-wire or four-wire. The two-wire implementation can be either shielded twisted pair, STP (**100Base-TX**) or fiber-optic cable (**100Base-FX**). The four-wire implementation is designed only for unshielded twist pair, UTP (**100Base-T4**). Table 3.2 is a summary of the Fast Ethernet implementations.

**Table 3.2** *Summary of Fast Ethernet implementations*

| *Characteristics* | *100Base-TX* | *100Base-FX* | *100Base-T4* |
|---|---|---|---|
| Media | STP | Fiber | UTP |
| Number of wires | 2 | 2 | 4 |
| Maximum length | 100 m | 100 m | 100 m |

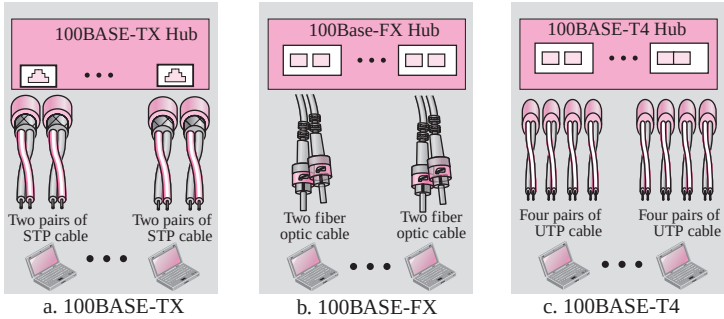
Figure 3.11 shows simplified diagrams of each implementation.

## Gigabit Ethernet

The need for an even higher data rate resulted in the design of the Gigabit Ethernet Protocol (1000 Mbps). The IEEE committee calls the Standard 802.3z. The goals of the Gigabit Ethernet design can be summarized as follows:

1. Upgrade the data rate to 1 Gbps.
2. Make it compatible with Standard or Fast Ethernet.

**Figure 3.11** *Fast Ethernet Implementation*




a. 100BASE-TX b. 100BASE-FX c. 100BASE-T4

3. Use the same 48-bit address.
4. Use the same frame format.
5. Keep the same minimum and maximum frame lengths.
6. To support autonegotiation as defined in Fast Ethernet.

### *MAC Sublayer*

A main consideration in the evolution of Ethernet was to keep the MAC sublayer untouched. However, to achieve a data rate of 1 Gbps, this was no longer possible. Gigabit Ethernet has two distinctive approaches for medium access: half-duplex and full-duplex. Almost all implementations of Gigabit Ethernet follow the full-duplex approach. However, we briefly discuss the half-duplex approach to show that Gigabit Ethernet can be compatible with the previous generations.

**Full-Duplex Mode** In full-duplex mode, there is a central switch connected to all computers or other switches. In this mode, each switch has buffers for each input port in which data are stored until they are transmitted. There is no collision in this mode. This means that CSMA/CD is not used. Lack of collision implies that the maximum length of the cable is determined by the signal attenuation in the cable, not by the collision detection process.

**In the full-duplex mode of Gigabit Ethernet, there is no collision; the maximum length of the cable is determined by the signal attenuation in the cable.**

**Half-Duplex Mode** Gigabit Ethernet can also be used in half-duplex mode, although it is rare. In this case, a switch can be replaced by a hub, which acts as the common cable in which a collision might occur. The half-duplex approach uses CSMA/CD. However, as we saw before, the maximum length of the network in this approach is totally dependent on the minimum frame size. Three solutions have been defined: traditional, carrier extension, and frame bursting.

- **Traditional.** In the traditional approach, we keep the minimum length of the frame as in traditional Ethernet (512 bits). However, because the length of a bit is 1/100 shorter in Gigabit Ethernet than in 10-Mbps Ethernet, the maximum length of the network is 25 m. This length may be suitable if all the stations are in one room, but it may not even be long enough to connect the computers in one single office.
- **Carrier Extension.** To allow for a longer network, we increase the minimum frame length. The **carrier extension** approach defines the minimum length of a frame as 512 bytes (4096 bits). This means that the minimum length is 8 times longer. This method forces a station to add extension bits (padding) to any frame that is less than 4096 bits. In this way, the maximum length of the network can be increased 8 times to a length of 200 m. This allows a length of 100 m from the hub to the station.
- **Frame Bursting**. Carrier extension is very inefficient if we have a series of short frames to send; each frame carries redundant data. To improve efficiency, **frame bursting** was proposed. Instead of adding an extension to each frame, multiple frames are sent. However, to make these multiple frames look like one frame, padding is added between the frames (the same as that used for the carrier extension method) so that the channel is not idle. In other words, the method deceives other stations into thinking that a very large frame has been transmitted.

### *Implementation*

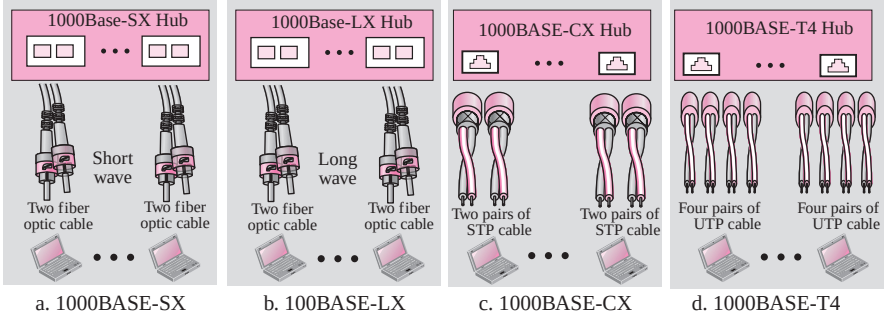
Table 3.3 is a summary of the Gigabit Ethernet implementations.

**Table 3.3** *Summary of Gigabit Ethernet implementations*

| *Characteristics* | *1000Base-SX* | *1000Base-LX* | *1000Base-CX* | *1000Base-T4* |
|---|---|---|---|---|
| Media | Fiber short-wave | Fiber long-wave | STP | Cat 5 UTP |
| Number of wires | 2 | 2 | 2 | 4 |
| Maximum length | 550 m | 5000 m | 25 m | 100 m |

Figure 3.12 shows the simplified diagrams for Gigabit Ethernet.

**Figure 3.12** *Gigabit Ethernet implementation*




a. 1000BASE-SX b. 100BASE-LX c. 1000BASE-CX d. 1000BASE-T4

### Ten-Gigabit Ethernet

The IEEE committee created Ten-Gigabit Ethernet and called it Standard 802.3ae. The goals of the Ten-Gigabit Ethernet design can be summarized as follows:

1. Upgrade the data rate to 10 Gbps.
2. Make it compatible with Standard, Fast, and Gigabit Ethernet.
3. Use the same 48-bit address.
4. Use the same frame format.
5. Keep the same minimum and maximum frame lengths.
6. Allow the interconnection of existing LANs into a metropolitan area network (MAN) or a wide area network (WAN).
7. Make Ethernet compatible with technologies such as Frame Relay and ATM.

#### *Implementation*

Ten-Gigabit Ethernet operates only in full duplex mode, which means there is no need for contention; CSMA/CD is not used in Ten-Gigabit Ethernet. Three implementations are the most common: 10GBase-S, 10GBase-L, and 10GBase-E. Table 3.4 shows a summary of the Ten-Gigabit Ethernet implementation.

**Table 3.4** *Ten-Gigabit Ethernet Implementation*

| *Characteristics* | *10GBase-S* | *10GBase-L* | *10GBase-E* |
|---|---|---|---|
| Media | multi-mode fiber | single-mode fiber | single-mode fiber |
| Number of wires | 2 | 2 | 2 |
| Maximum length | 300 m | 10,000 m | 40,000 m |

## 3.2 WIRELESS LANS

Wireless communication is one of the fastest-growing technologies. The demand for connecting devices without the use of cables is increasing everywhere. **Wireless LANs** can be found on college campuses, in office buildings, and in many public areas. In this section, we concentrate on two wireless technologies for LANs: IEEE 802.11 wireless LANs, sometimes called wireless Ethernet, and Bluetooth, a technology for small wireless LANs.

### IEEE 802.11

IEEE has defined the specifications for a wireless LAN, called **IEEE 802.11,** which covers the physical and data link layers.

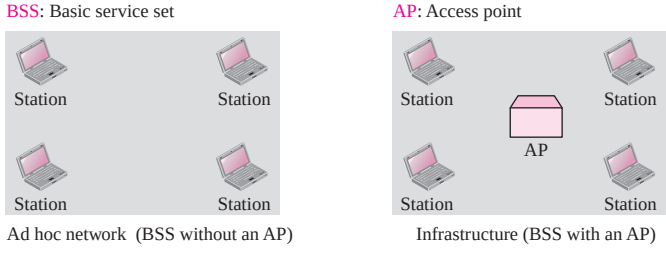
#### *Architecture*

The standard defines two kinds of services: the basic service set (BSS) and the extended service set (ESS).

**Basic Service Set** IEEE 802.11 defines the **basic service set (BSS)** as the building block of a wireless LAN. A basic service set is made of stationary or mobile wireless stations and an optional central base station, known as the **access point (AP).** Figure 3.13 shows two sets in this standard. The BSS without an AP is a stand-alone network and

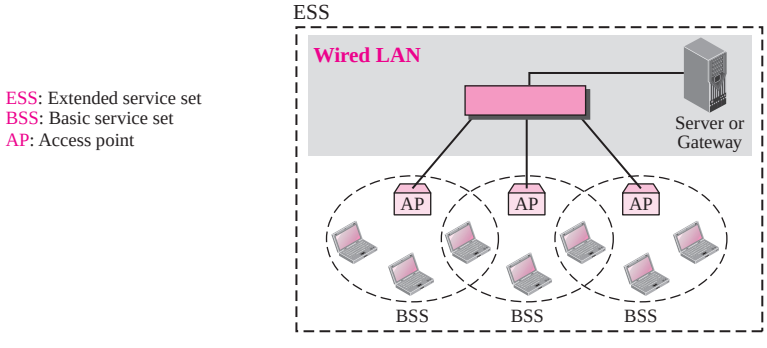
cannot send data to other BSSs. It is called an *ad hoc architecture*. In this architecture, stations can form a network without the need of an AP; they can locate one another and agree to be part of a BSS. A BSS with an AP is sometimes referred to as an *infrastructure* network.

**Figure 3.13** *Basic service sets (BSSs)*




**Extended Service Set** An **extended service set (ESS)** is made up of two or more BSSs with APs. In this case, the BSSs are connected through a *distribution system*, which is usually a wired LAN. The distribution system connects the APs in the BSSs. IEEE 802.11 does not restrict the distribution system; it can be any IEEE LAN such as an Ethernet. Note that the extended service set uses two types of stations: mobile and-stationary. The mobile stations are normal stations inside a BSS. The stationary stations are AP stations that are part of a wired LAN. Figure 3.14 shows an ESS.

**Figure 3.14** *Extended service sets (ESSs)*




When BSSs are connected, the stations within reach of one another can communicate without the use of an AP. However, communication between two stations in two different BSSs usually occurs via two APs.

### *Station Types*

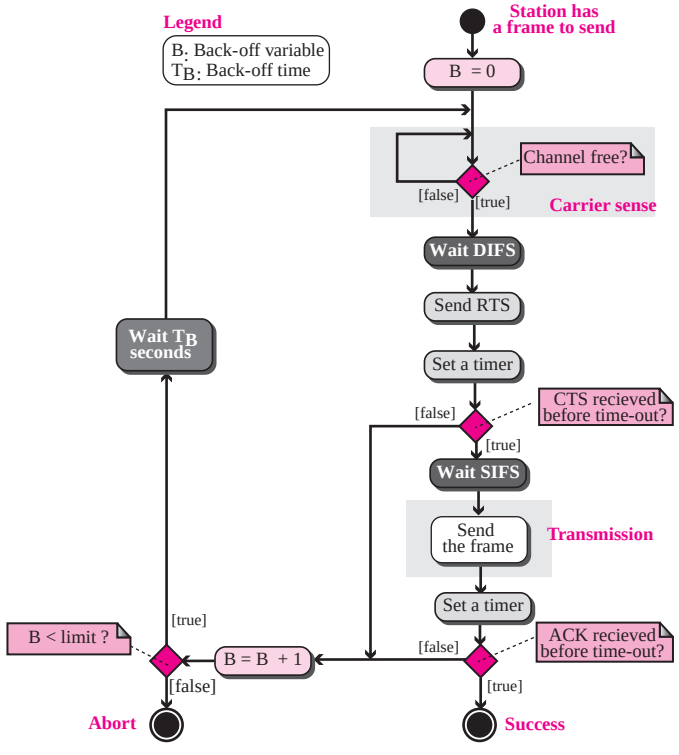
IEEE 802.11 defines three types of stations based on their mobility in a wireless LAN: **no-transition, BSS-transition,** and **ESS-transition mobility.** A station with no-transition

mobility is either stationary (not moving) or moving only inside a BSS. A station with BSS-transition mobility can move from one BSS to another, but the movement is confined inside one ESS. A station with ESS-transition mobility can move from one ESS to another.

## MAC Sublayer

There are two different MAC sublayers in this protocol, however; the one that is used most of the time is based on CSMA/CA (**carrier sense multiple access with collision avoidance**). Figure 3.15 shows the flow diagram.

**Figure 3.15** *CSMA/CA flow diagram*



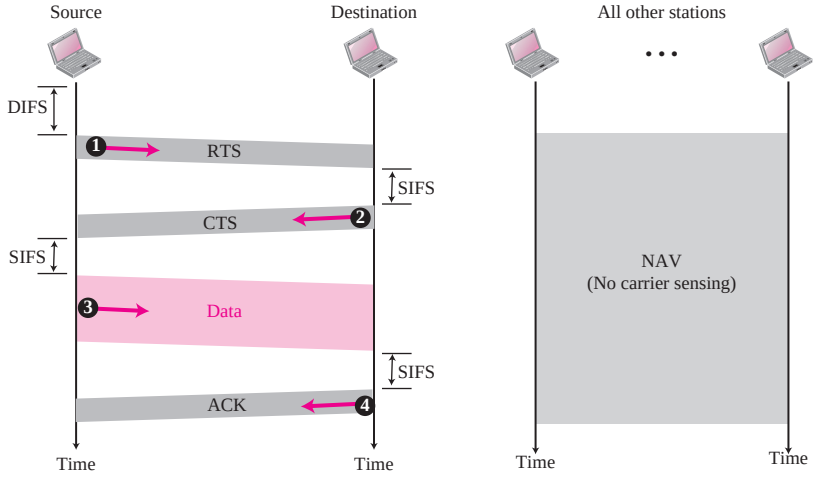

Wireless LANs cannot implement CSMA/CD for three reasons:

1. For collision detection a station must be able to send data and receive collision signals at the same time. This can mean costly stations and increased bandwidth requirements.
2. Collision may not be detected because of the hidden station problem. We will discuss this problem later in the chapter.
3. The distance between stations can be great. Signal fading could prevent a station at one end from hearing a collision at the other end.

### *Frame Exchange Time Line*

Figure 3.16 shows the exchange of data and control frames in time.

**Figure 3.16** *CSMA/CA and NAV*




1. Before sending a frame, the source station senses the medium by checking the energy level at the carrier frequency.
   a. The channel uses a persistence strategy with back-off until the channel is idle.
   b. After the station is found to be idle, the station waits for a period of time called the **distributed interframe space (DIFS);** then the station sends a control frame called the request to send (RTS).
2. After receiving the RTS and waiting a period of time called the **short interframe space (SIFS),** the destination station sends a control frame, called the clear to send (CTS), to the source station. This control frame indicates that the destination station is ready to receive data.
3. The source station sends data after waiting an amount of time equal to SIFS.
4. The destination station, after waiting an amount of time equal to SIFS, sends an acknowledgment to show that the frame has been received. Acknowledgment is needed in this protocol because the station does not have any means to check for the successful arrival of its data at the destination. On the other hand, the lack of collision in CSMA/CD is a kind of indication to the source that data have arrived.

**Network Allocation Vector** How do other stations defer sending their data if one station acquires access? In other words, how is the *collision avoidance* aspect of this protocol accomplished? The key is a feature called NAV.

When a station sends an RTS frame, it includes the duration of time that it needs to occupy the channel. The stations that are affected by this transmission create a timer called a **network allocation vector (NAV)** that shows how much time must pass before these stations are allowed to check the channel for idleness. Each time a station

accesses the system and sends an RTS frame, other stations start their NAV. In other words, each station, before sensing the physical medium to see if it is idle, first checks its NAV to see if it has expired. Figure 3.16 also shows the idea of NAV.

What happens if there is collision during the time when RTS or CTS control frames are in transition, often called the **handshaking period**? Two or more stations may try to send RTS frames at the same time. These control frames may collide. However, because there is no mechanism for collision detection, the sender assumes there has been a collision if it has not received a CTS frame from the receiver. The back-off strategy is employed, and the sender tries again.

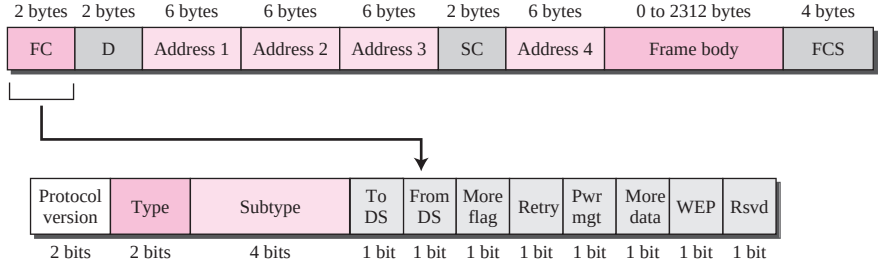
### *Fragmentation*

The wireless environment is very noisy; a corrupt frame has to be retransmitted. The protocol, therefore, recommends fragmentation—the division of a large frame into smaller ones. It is more efficient to resend a small frame than a large one.

### *Frame Format*

The MAC layer frame consists of nine fields, as shown in Figure 3.17.

**Figure 3.17** *Frame format*




- **Frame control (FC).** The FC field is 2 bytes long and defines the type of frame and some control information. Table 3.5 describes the subfields. We will discuss each frame type later in this chapter.

**Table 3.5** *Subfields in FC field*

| *Field* | *Explanation* |
|---|---|
| Version | Current version is 0 |
| Type | Type of information: management (00), control (01), or data (10) |
| Subtype | Subtype of each type (see Table 3.6) |
| To DS | Defined later |
| From DS | Defined later |
| More flag | When set to 1, means more fragments |
| Retry | When set to 1, means retransmitted frame |
| Pwr mgt | When set to 1, means station is in power management mode |
| More data | When set to 1, means station has more data to send |
| WEP | Wired equivalent privacy (encryption implemented) |
| Rsvd | Reserved |

- **D.** In all frame types except one, this field defines the duration of the transmission that is used to set the value of NAV. In one control frame, this field defines the ID of the frame.
- **Addresses.** There are four address fields, each 6 bytes long. The meaning of each address field depends on the value of the *To DS* and *From DS* subfields and will be discussed later.
- **Sequence control.** This field defines the sequence number of the frame to be used in flow control.
- **Frame body.** This field, which can be between 0 and 2312 bytes, contains information based on the type and the subtype defined in the FC field.
- **FCS.** The FCS field is 4 bytes long and contains a CRC-32 error detection sequence.

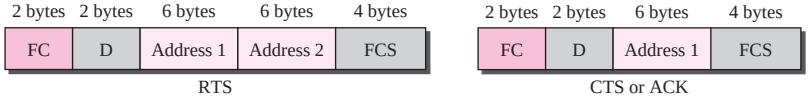
### *Frame Types*

A wireless LAN defined by IEEE 802.11 has three categories of frames: management frames, control frames, and data frames.

**Management Frames** Management frames are used for the initial communication between stations and access points.

**Control Frames** Control frames are used for accessing the channel and acknowledging frames. Figure 3.18 shows the format.

**Figure 3.18** *Control frames*




For control frames the value of the type field is 01; the values of the subtype fields for frames we have discussed are shown in Table 3.6.

**Table 3.6** *Values of subfields in control frames*

| *Subtype* | *Meaning* |
|---|---|
| 1011 | Request to send (RTS) |
| 1100 | Clear to send (CTS) |
| 1101 | Acknowledgment (ACK) |

**Data Frames** Data frames are used for carrying data and control information.

## Addressing Mechanism

The IEEE 802.11 addressing mechanism specifies four cases, defined by the value of the two flags in the FC field, *To DS* and *From DS*. Each flag can be either 0 or 1, resulting in four different situations. The interpretation of the four addresses (address 1 to address 4) in the MAC frame depends on the value of these flags, as shown in Table 3.7.

**Table 3.7** *Addresses*

| *To DS* | *From DS* | *Address 1* | *Address 2* | *Address 3* | *Address 4* |
|---|---|---|---|---|---|
| 0 | 0 | Destination | Source | BSS ID | N/A |
| 0 | 1 | Destination | Sending AP | Source | N/A |
| 1 | 0 | Receiving AP | Source | Destination | N/A |
| 1 | 1 | Receiving AP | Sending AP | Destination | Source |

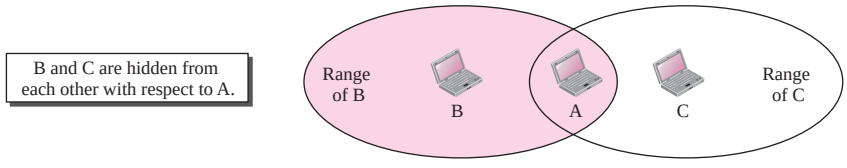
Note that address 1 is always the address of the next device. Address 2 is always the address of the previous device. Address 3 is the address of the final destination station if it is not defined by address 1. Address 4 is the address of the original source station if it is not the same as address 2.

### *Hidden and Exposed Station Problems*

We referred to hidden and exposed station problems in the previous section. It is time now to discuss these problems and their effects.

**Hidden Station Problem** Figure 3.19 shows an example of the hidden station problem. Station B has a transmission range shown by the left oval (sphere in space); every station in this range can hear any signal transmitted by station B. Station C has a transmission range shown by the right oval (sphere in space); every station located in this range can hear any signal transmitted by C. Station C is outside the transmission range of B; likewise, station B is outside the transmission range of C. Station A, however, is in the area covered by both B and C; it can hear any signal transmitted by B or C.

**Figure 3.19** *Hidden station problem*



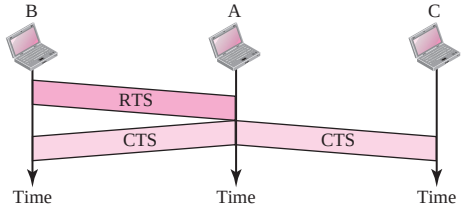

Assume that station B is sending data to station A. In the middle of this transmission, station C also has data to send to station A. However, station C is out of B's range and transmissions from B cannot reach C. Therefore C thinks the medium is free. Station C sends its data to A, which results in a collision at A because this station is receiving data from both B and C. In this case, we say that stations B and C are hidden from each other with respect to A. Hidden stations can reduce the capacity of the network because of the possibility of collision.

The solution to the hidden station problem is the use of the handshake frames (RTS and CTS) that we discussed earlier. Figure 3.20 shows that the RTS message from B reaches A, but not C. However, because both B and C are within the range of A, the CTS message, which contains the duration of data transmission from B to A reaches C.

Station C knows that some hidden station is using the channel and refrains from transmitting until that duration is over.

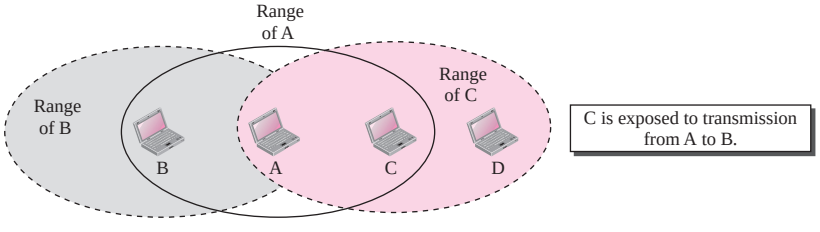
**The CTS frame in CSMA/CA handshake can prevent collision from a hidden station.**

**Figure 3.20** *Use of handshaking to prevent hidden station problem*




**Exposed Station Problem** Now consider a situation that is the inverse of the previous one: the exposed station problem. In this problem a station refrains from using a channel when it is, in fact, available. In Figure 3.21, station A is transmitting to station B. Station C has some data to send to station D, which can be sent without interfering with the transmission from A to B. However, station C is exposed to transmission from A; it hears what A is sending and thus refrains from sending. In other words, C is too conservative and wastes the capacity of the channel.

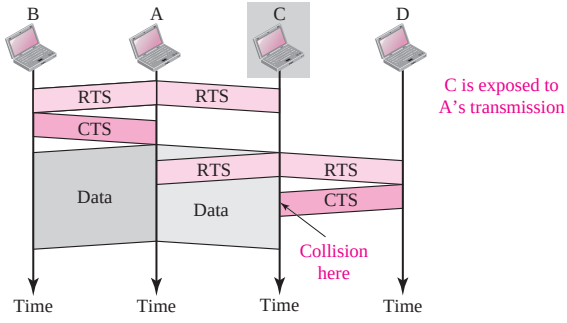
**Figure 3.21** *Exposed station problem*




The handshaking messages RTS and CTS cannot help in this case, despite what we might think. Figure 3.22 shows the situation.

Station C hears the RTS from A, but does not hear the CTS from B. Station C, after hearing the RTS from A, can wait for a time so that the CTS from B reaches A; it then sends an RTS to D to show that it needs to communicate with D. Both stations B and A may hear this RTS, but station A is in the sending state, not the receiving state. Station B, however, responds with a CTS. The problem is here. If station A has started sending its data, station C cannot hear the CTS from station D because of the collision; it cannot send its data to D. It remains exposed until A finishes sending its data.

**Figure 3.22** *Use of handshaking in exposed station problem*




## Bluetooth

**Bluetooth** is a wireless LAN technology designed to connect devices of different functions such as telephones, notebooks, computers (desktop and laptop), cameras, printers, coffee makers, and so on. A Bluetooth LAN is an ad hoc network, which means that the network is formed spontaneously; the devices, sometimes called gadgets, find each other and make a network called a piconet. A Bluetooth LAN can even be connected to the Internet if one of the gadgets has this capability. A Bluetooth LAN, by nature, cannot be large. If there are many gadgets that try to connect, there is chaos.

Bluetooth technology has several applications. Peripheral devices such as a wireless mouse or keyboard can communicate with the computer through this technology. Monitoring devices can communicate with sensor devices in a small health care center. Home security devices can use this technology to connect different sensors to the main security controller. Conference attendees can synchronize their laptop computers at a conference.

Bluetooth was originally started as a project by the Ericsson Company. It is named for Harald Blaatand, the king of Denmark (940–981) who united Denmark and Norway. *Blaatand* translates to *Bluetooth* in English.

Today, Bluetooth technology is the implementation of a protocol defined by the IEEE 802.15 standard. The standard defines a wireless personal area network (PAN) operable in an area the size of a room or a hall.

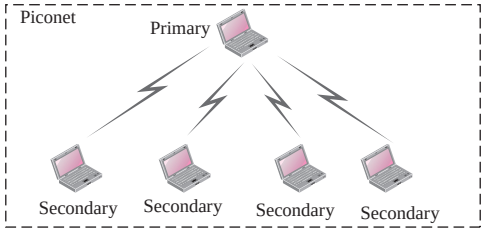
### *Architecture*

Bluetooth defines two types of networks: piconet and scatternet.

**Piconets** A Bluetooth network is called a **piconet,** or a small net. A piconet can have up to eight stations, one of which is called the **primary;** the rest are called **secondaries.** All the secondary stations synchronize their clocks and hopping sequence with the primary. Note that a piconet can have only one primary station. The communication between the primary and the secondary can be one-to-one or one-to-many. Figure 3.23 shows a piconet.

Although a piconet can have a maximum of seven secondaries, an additional eight secondaries can be in the *parked state*. A secondary in a parked state is synchronized

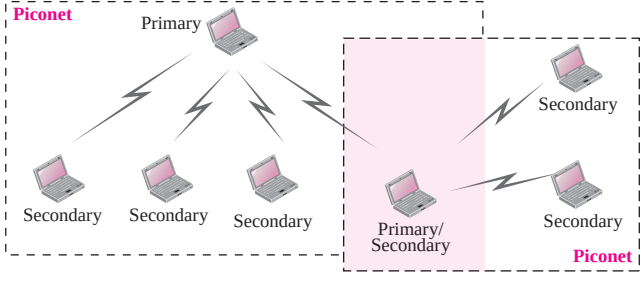
**Figure 3.23** *Piconet*




with the primary, but cannot take part in communication until it is moved from the parked state. Because only eight stations can be active in a piconet, activating a station from the parked state means that an active station must go to the parked state.

**Scatternet** Piconets can be combined to form what is called a **scatternet.** A secondary station in one piconet can be the primary in another piconet. This station can receive messages from the primary in the first piconet (as a secondary) and, acting as a primary, deliver them to secondaries in the second piconet. A station can be a member of two piconets. Figure 3.24 illustrates a scatternet.

**Figure 3.24** *Scatternet*




### *Bluetooth Devices*

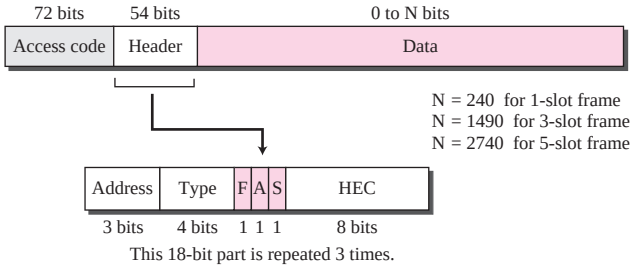
A Bluetooth device has a built-in short-range radio transmitter. The current data rate is 1 Mbps with a 2.4-GHz bandwidth. This means that there is a possibility of interference between the IEEE 802.11b wireless LANs and Bluetooth LANs.

### *Frame Format*

A frame in the baseband layer can be one of three types: one-slot, three-slot, or five-slot. A slot, as we said before, is 625 μs. However, in a one-slot frame exchange, 259 μs is needed for hopping and control mechanisms. This means that a one-slot frame can last only 625 – 259, or 366 μs. With a 1-MHz bandwidth and 1 bit/Hz, the size of a one-slot frame is 366 bits.

A three-slot frame occupies three slots. However, since 259 μs is used for hopping, the length of the frame is $3 \times 625 - 259 = 1616$ μs or 1616 bits. A device that uses a three-slot frame remains at the same hop (at the same carrier frequency) for three slots. Even though only one hop number is used, three hop numbers are consumed. That means the hop number for each frame is equal to the first slot of the frame. A five-slot frame also uses 259 bits for hopping, which means that the length of the frame is $5 \times 625 - 259 = 2866$ bits. Figure 3.25 shows the format of the three frame types.

**Figure 3.25** *Frame format types*




The following describes each field:

- **Access code.** This 72-bit field normally contains synchronization bits and the identifier of the primary to distinguish the frame of one piconet from another.
- **Header.** This 54-bit field is a repeated 18-bit pattern. Each pattern has the following subfields:
  - **a. Address.** The 3-bit address subfield can define up to seven secondaries (1 to 7). If the address is zero, it is used for broadcast communication from the primary to all secondaries.
  - **b. Type.** The 4-bit type subfield defines the type of data coming from the upper layers. We discuss these types later.
  - **c. F.** This 1-bit subfield is for flow control. When set (1), it indicates that the device is unable to receive more frames (buffer is full).
  - **d. A.** This 1-bit subfield is for acknowledgment. Bluetooth uses stop-and-wait ARQ; 1 bit is sufficient for acknowledgment.
  - **e. S.** This 1-bit subfield holds a sequence number. Bluetooth uses stop-and-wait ARQ; 1 bit is sufficient for sequence numbering.
  - **f. HEC.** The 8-bit header error correction subfield is a checksum to detect errors in each 18-bit header section.

  The header has three identical 18-bit sections. The receiver compares these three sections, bit by bit. If each of the corresponding bits is the same, the bit is accepted; if not, the majority opinion rules. This is a form of forward error correction (for the header only). This double error control is needed because the nature of the communication, via air, is very noisy. Note that there is no retransmission in this sublayer.

- **Data.** This subfield can be 0 to 2740 bits long. It contains data or control information coming from the upper layers.

**a.** The sending station, after sensing that the medium is idle, sends a special small frame called request to send (RTS). In this message, the sender defines the total time it needs the medium.

**b.** The receiver acknowledges the request (broadcast to all stations) by sending a small packet called clear to send (CTS).

**c.** The sender sends the data frame.

**d.** The receiver acknowledges the receipt of data.

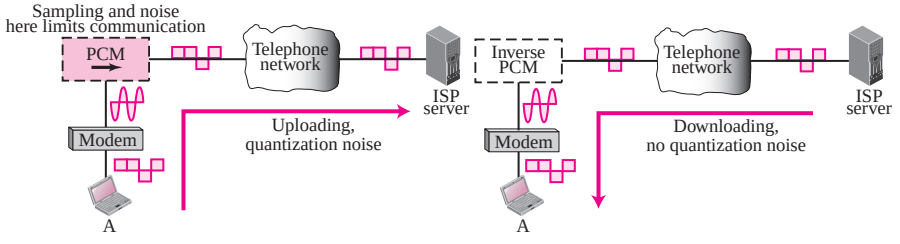
## 3.3 POINT-TO-POINT WANS

A second type of network we encounter in the Internet is the point-to-point wide area network. A point-to-point WAN connects two remote devices using a line available from a public network such as a telephone network. We discuss traditional modem technology, DSL line, cable modem, T-lines, and SONET.

### 56K Modems

People still use traditional modems to upload data to the Internet and download data from the Internet, as shown in Figure 3.26.

**Figure 3.26** *56K modem*




In **uploading,** the analog signal must be sampled at the switching station, which means the data rate in uploading is limited to 33.6 kbps. However, there is no sampling in **downloading.** The signal is not affected by quantization noise and not subject to the Shannon capacity limitation. The maximum data rate in the uploading direction is 33.6 kbps, but the data rate in the downloading direction is 56 kbps.

One may wonder why 56 kbps. The telephone companies sample voice 8000 times per second with 8 bits per sample. One of the bits in each sample is used for control purposes, which means each sample is 7 bits. The rate is therefore $8000 \times 7$, or 56,000 bps or 56 kbps.

The **V.90** and **V.92** standard modems operate at 56 kbps to connect a host to the Internet.

## DSL Technology

After traditional modems reached their peak data rate, telephone companies developed another technology, DSL, to provide higher-speed access to the Internet. **Digital subscriber line (DSL)** technology is one of the most promising for supporting high-speed digital communication over the existing local loops (telephone line). DSL technology is a set of technologies, each differing in the first letter (ADSL, VDSL, HDSL, and SDSL). The set is often referred to as *x*DSL, where *x* can be replaced by A, V, H, or S.

### *ADSL*

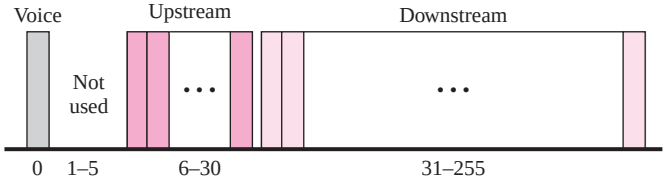
The first technology in the set is **asymmetric DSL (ADSL).** ADSL, like a 56K modem, provides higher speed (bit rate) in the downstream direction (from the Internet to the resident) than in the upstream direction (from the resident to the Internet). That is the reason it is called asymmetric. Unlike the asymmetry in 56K modems, the designers of ADSL specifically divided the available bandwidth of the local loop unevenly for the residential customer. The service is not suitable for business customers who need a large bandwidth in both directions.

**ADSL is an asymmetric communication technology designed for residential users; it is not suitable for businesses.**

Figure 3.27 shows how the bandwidth is divided:

- **Voice.** Channel 0 is reserved for voice communication.
- **Idle.** Channels 1 to 5 are not used, to allow a gap between voice and data communication.
- **Upstream data and control.** Channels 6 to 30 (25 channels) are used for upstream data transfer and control. One channel is for control, and 24 channels are for data transfer. If there are 24 channels, each using 4 kHz (out of 4.312 kHz available) with 15 bits per Hz, we have $24 \times 4000 \times 15$, or 1.44-Mbps bandwidth, in the upstream direction.
- **Downstream data and control.** Channels 31 to 255 (225 channels) are used for downstream data transfer and control. One channel is for control, and 224 channels are for data. If there are 224 channels, we can achieve up to $224 \times 4000 \times 15$, or 13.4 Mbps.

**Figure 3.27** *Bandwidth division*

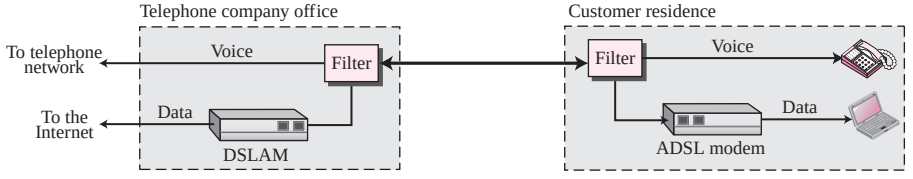


Because of the high signal/noise ratio, the actual bit rate is much lower than the above-mentioned rates. The bit rates are as follows:

| Upstream: 64 kbps to 1 Mbps | Downstream: 500 kbps to 8 Mbps |
|---|---|

Figure 3.28 shows an ADSL modem installed at a customer's site. The local loop connects to the filter which separates voice and data communication. The ADSL modem modulates the data and creates downstream and upstream channels.

**Figure 3.28** *ADSL and DSLAM*




At the telephone company site, the situation is different. Instead of an ADSL modem, a device called a **digital subscriber line access multiplexer (DSLAM)** is installed that functions similarly to an ADSL modem. In addition, it packetizes the data to be sent to the Internet. Figure 3.28 shows the configuration.

### *Other DSL Technologies*

ADSL provides asymmetric communication. The downstream bit rate is much higher than the upstream bit rate. Although this feature meets the needs of most residential subscribers, it is not suitable for businesses that send and receive data in large volumes in both directions. The **symmetric digital subscriber line (SDSL)** is designed for these types of businesses. It divides the available bandwidth equally between the downstream and upstream directions.

The **high bit rate digital subscriber line (HDSL)** was designed as an alternative to the T-1 line (1.544 Mbps). The T-1 line (discussed later) uses alternate mark inversion (AMI) encoding, which is very susceptible to attenuation at high frequencies. This limits the length of a T-1 line to 1 km. For longer distances, a repeater is necessary, which means increased costs.

The **very high bit rate digital subscriber line (VDSL),** an alternative approach that is similar to ADSL, uses coaxial, fiber-optic, or twisted-pair cable for short distances (300 to 1800 m). The modulating technique is discrete multitone technique (DMT) with a bit rate of 50 to 55 Mbps downstream and 1.5 to 2.5 Mbps upstream.

## Cable Modem

Cable companies are now competing with telephone companies for the residential customer who wants high-speed access to the Internet. DSL technology provides high-data-rate connections for residential subscribers over the local loop. However, DSL uses the existing unshielded twisted-pair cable, which is very susceptible to

interference. This imposes an upper limit on the data rate. Another solution is the use of the cable TV network.

### *Traditional Cable Networks*

**Cable TV** started to distribute broadcast video signals to locations with poor or no reception. It was called **community antenna TV (CATV)** because an antenna at the top of a high hill or building received the signals from the TV stations and distributed them, via coaxial cables, to the community.

The cable TV office, called the **head end,** receives video signals from broadcasting stations and feeds the signals into coaxial cables. The traditional cable TV system used coaxial cable end to end. Because of attenuation of the signals and the use of a large number of amplifiers, communication in the traditional network was unidirectional (one-way). Video signals were transmitted downstream, from the head end to the subscriber premises.

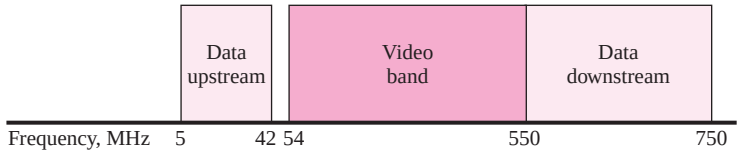
### *HFC Network*

The second generation of cable networks is called a **hybrid fiber-coaxial (HFC) network.** The network uses a combination of fiber-optic and coaxial cable. The transmission medium from the cable TV office to a box, called the **fiber node,** is optical fiber; from the fiber node through the neighborhood and into the house, the medium is still coaxial cable. One reason for moving from traditional to hybrid infrastructure is to make the cable network bidirectional (two-way).

### *Bandwidth*

Even in an HFC system, the last part of the network, from the fiber node to the subscriber premises, is still a coaxial cable. This coaxial cable has a bandwidth that ranges from 5 to 750 MHz (approximately). The cable company has divided this bandwidth into three bands: video, downstream data, and upstream data, as shown in Figure 3.29.

**Figure 3.29** *Cable bandwidth*




- **Video Band.** The downstream-only **video band** occupies frequencies from 54 to 550 MHz. Since each TV channel occupies 6 MHz, this can accommodate more than 80 channels.
- **Downstream Data Band.** The downstream data (from the Internet to the subscriber premises) occupies the upper band, from 550 to 750 MHz. This band is also divided into 6-MHz channels. The downstream data can be received at 30 Mbps. The standard specifies only 27 Mbps. However, since the cable modem is connected to the computer through a 10BASE-T cable, this limits the data rate to 10 Mbps.

- **Upstream Data Band.** The upstream data (from the subscriber premises to the Internet) occupies the lower band, from 5 to 42 MHz. This band is also divided into 6-MHz channels. The **upstream data band** uses lower frequencies that are more susceptible to noise and interference. Theoretically, downstream data can be sent at 12 Mbps (2 bits/Hz $\times$ 6 MHz). However, the data rate is usually less than 12 Mbps.

### *Sharing*

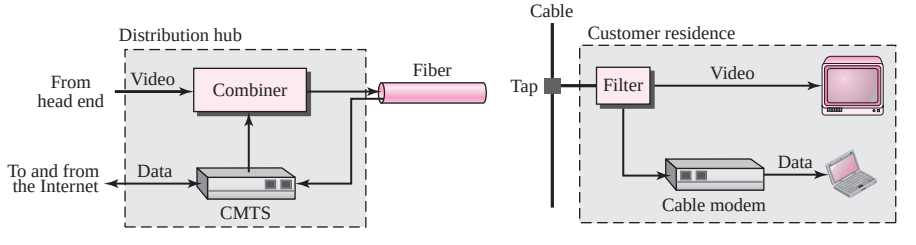
Both upstream and downstream bands are shared by the subscribers. The upstream data bandwidth is only 37 MHz. This means that there are only six 6-MHz channels available in the upstream direction. A subscriber needs to use one channel to send data in the upstream direction. The question is, How can six channels be shared in an area with 1000, 2000, or even 100,000 subscribers? The solution is time-sharing. The band is divided into channels; these channels must be shared between subscribers in the same neighborhood. The cable provider allocates one channel, statically or dynamically, for a group of subscribers. If one subscriber wants to send data, she or he contends for the channel with others who want access; the subscriber must wait until the channel is available. The situation is similar to CSMA discussed for Ethernet LANs.

We have a similar situation in the downstream direction. The downstream band has 33 channels of 6 MHz. A cable provider probably has more than 33 subscribers; therefore, each channel must be shared between a group of subscribers. However, the situation is different for the downstream direction; here we have a multicasting situation. If there are data for any of the subscribers in the group, the data are sent to that channel. Each subscriber is sent the data. But since each subscriber also has an address registered with the provider, the cable modem for the group matches the address carried with the data to the address assigned by the provider. If the address matches, the data are kept; otherwise, they are discarded.

### *Devices*

To use a cable network for data transmission, we need two key devices: a CM and a CMTS. The **cable modem (CM)** is installed on the subscriber premises. It is similar to an ADSL modem. Figure 3.30 shows its location. The **cable modem transmission system (CMTS)** is installed inside the distribution hub by the cable company. It receives data from the Internet and passes them to the combiner, which sends them to the subscriber. The CMTS also receives data from the subscriber and passes them to the Internet. Figure 3.30 shows the location of the CMTS.

**Figure 3.30** *Cable modem configurations*



## T Lines

**T lines** are standard digital telephone carriers originally designed to multiplex voice channels (after being digitized). Today, however, T lines can be used to carry data from a residence or an organization to the Internet. They can also be used to provide a physical link between nodes in a switched wide area network. T lines are commercially available in two data rates: T-1 and T-3 (see Table 3.8).

**Table 3.8** *T line rates*

| *Line* | *Rate (Mbps)* |
|---|---|
| T-1 | 1.544 |
| T-3 | 44.736 |

### *T-1 Line*

The data rate of a **T-1 line** is 1.544 Mbps. Twenty-four voice channels are sampled, with each sample digitized to 8 bits. An extra bit is added to provide synchronization. This makes the frame 193 bits in length. By sending 8000 frames per second, we get a data rate of 1.544 Mbps. When we use a T-1 line to connect to the Internet, we can use all or part of the capacity of the line to send digital data.

### *T-3 Line*

A **T-3 line** has a data rate of 44.736 Mbps. It is equivalent to 28 T-1 lines. Many subscribers may not need the entire capacity of a T line. To accommodate these customers, the telephone companies have developed fractional T line services, which allow several subscribers to share one line by multiplexing their transmissions.

## SONET

The high bandwidths of fiber-optic cable are suitable for today's highest data rate technologies (such as video conferencing) and for carrying large numbers of lower-rate technologies at the same time. ANSI created a set of standards called **Synchronous Optical Network (SONET)** to handle the use of fiber-optic cables. It defines a high-speed data carrier.

SONET first defines a set of electrical signals called **synchronous transport signals (STSs).** It then converts these signals to optical signals called **optical carriers (OCs).** The optical signals are transmitted at 8000 frames per second.

Table 3.9 shows the data rates for STSs and OCs. Note that the lowest level in this hierarchy has a data rate of 51.840 Mbps, which is greater than that of a T-3 line (44.736 Mbps).

**Table 3.9** *SONET rates*

| *STS* | *OC* | *Rate (Mbps)* | *STS* | *OC* | *Rate (Mbps)* |
|---|---|---|---|---|---|
| STS-1 | OC-1 | 51.840 | STS-24 | OC-24 | 1244.160 |
| STS-3 | OC-3 | 155.520 | STS-36 | OC-36 | 1866.230 |
| STS-9 | OC-9 | 466.560 | STS-48 | OC-48 | 2488.320 |
| STS-12 | OC-12 | 622.080 | STS-96 | OC-96 | 4976.640 |
| STS-18 | OC-18 | 933.120 | STS-192 | OC-192 | 9953.280 |

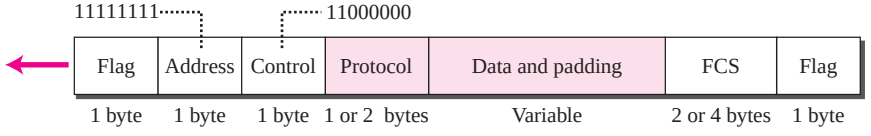
## PPP

The telephone line or cable companies provide a physical link, but to control and manage the transfer of data, there is a need for a special protocol. The **Point-to-Point Protocol (PPP**) was designed to respond to this need.

### *PPP Layers*

PPP has only physical and data link layers. No specific protocol is defined for the physical layer by PPP. Instead, it is left to the implementer to use whatever is available. PPP supports any of the protocols recognized by ANSI. At the data link layer, PPP defines the format of a frame and the protocol that are used for controlling the link and transporting user data. The format of a PPP frame is shown in Figure 3.31.

**Figure 3.31** *PPP frame*




The descriptions of the fields are as follows:

1. **Flag field.** The flag field identifies the boundaries of a PPP frame. Its value is 01111110.
2. **Address field.** Because PPP is used for a point-to-point connection, it uses the broadcast address used in most LANs, 11111111, to avoid a data link address in the protocol.
3. **Control field.** The control field is assigned the value 11000000 to show that, as in most LANs, the frame has no sequence number; each frame is independent.
4. **Protocol field.** The protocol field defines the type of data being carried in the data field: user data or other information.
5. **Data field.** This field carries either user data or other information.
6. **FCS.** The frame check sequence field is simply a 2-byte or 4-byte CRC used for error detection.

### *Link Control Protocol (LCP)*

The **Link Control Protocol (LCP)** is responsible for establishment, maintenance, and termination of the link. When the data field of a frame is carrying data related to this protocol, it means that PPP is handling the link; it does not carry data.

### *Network Control Protocol (NCP)*

The **Network Control Protocol (NCP)** has been defined to give flexibility to PPP. PPP can carry data from different network protocols, including IP. After establishment of the link, PPP can carry IP packets in its data field.

### *PPPoE*

PPP was designed to connect a single user to the Internet via a conventional modem and a telephone line. Today, DSL, cable modem, and wireless technology allow a group of users, on an Ethernet LAN, to access the Internet through a single physical line. In other words, the hosts connected to the LAN can share one single physical line to access the Internet. **PPP over Ethernet (PPPoE)** is a new protocol that uses a discovery technique to find the Ethernet address of the host to be connected to the Internet. After address discovery, a regular PPP session can be used to provide the connection.

## 3.4 SWITCHED WANS

The backbone networks in the Internet can be switched WANs. A switched WAN is a wide area network that covers a large area (a state or a country) and provides access at several points to the users. Inside the network, there is a mesh of point-to-point networks that connects switches. The switches, multiple port connectors, allow the connection of several inputs and outputs.

Switched WAN technology differs from LAN technology in many ways. First, instead of a star topology, switches are used to create multiple paths. Second, LAN technology is considered a connectionless technology; there is no relationship between packets sent by a sender to a receiver. Switched WAN technology, on the other hand, is a connection-oriented technology. Before a sender can send a packet, a connection must be established between the sender and the receiver. After the connection is established, it is assigned an identifier (sometimes called a label) used during the transmission. The connection is formally terminated when the transmission is over. The connection identifier is used instead of the source and destination addresses in LAN technology.

### X.25

The **X.25** protocol, introduced in the 1970s, was the first switched WAN to become popular both in Europe and the United States. It was mostly used as a public network to connect individual computers or LANs. It provides an end-to-end service.

Although X.25 was used as the WAN to carry IP packets from one part of the world to another, there was always a conflict between IP and X.25. IP is a third-(network) layer protocol. An IP packet is supposed to be carried by a frame at the second (data link) layer. X.25, which was designed before the Internet, is a three-layer protocol; it has its own network layer. IP packets had to be encapsulated in an X.25 network-layer packet to be carried from one side of the network to another. This is analogous to a person who has a car but has to load it in a truck to go from one point to another.

Another problem with X.25 is that it was designed at a time when transmission media were not very reliable (no use of optical fibers). For this reason, X.25 performs extensive error control. This makes transmission very slow and is not popular given the ever increasing demand for speed.

## Frame Relay

The **Frame Relay** protocol, a switched technology that provides low-level (physical and data link layers) service, was designed to replace X.25. Frame Relay has some advantages over X.25:

- **High Data Rate.** Although Frame Relay was originally designed to provide a 1.544-Mbps data rate (equivalent to a T-1 line), today most implementations can handle up to 44.736 Mbps (equivalent to a T-3 line).
- **Bursty Data.** Some services offered by wide area network providers assume that the user has a fixed-rate need. For example, a T-1 line is designed for a user who wants to use the line at a consistent 1.544 Mbps. This type of service is not suitable for the many users today who need to send **bursty data** (non-fixed-rate data). For example, a user may want to send data at 6 Mbps for 2 seconds, 0 Mbps (nothing) for 7 seconds, and 3.44 Mbps for 1 second for a total of 15.44 Mb during a period of 10 seconds. Although the average data rate is still 1.544 Mbps, the T-1 line cannot fulfill this type of demand because it is designed for fixed-rate data, not bursty data. Bursty data requires what is called **bandwidth on demand.** The user needs different bandwidth allocations at different times. Frame Relay accepts bursty data. A user is granted an average data rate that can be exceeded when needed.
- **Less Overhead Due to Improved Transmission Media.** The quality of transmission media has improved tremendously since the last decade. They are more reliable and less error prone. There is no need to have a WAN that spends time and resources checking and double-checking potential errors. X.25 provides extensive error checking and flow control. Frame Relay does not provide error checking or require acknowledgment in the data link layer. Instead, all error checking is left to the protocols at the network and transport layers that use the services of Frame Relay.

## ATM

**Asynchronous Transfer Mode (ATM)** is the *cell relay* protocol designed by the ATM Forum and adopted by the ITU-T.

### *Design Goals*

Among the challenges faced by the designers of ATM, six stand out. First and foremost is the need for a transmission system to optimize the use of high-data-rate transmission media, in particular optical fiber. Second is the need for a system that can interface with existing systems, such as the various packet networks, and provide wide area interconnectivity between them without lowering their effectiveness or requiring their replacement. Third is the need for a design that can be implemented inexpensively so that cost would not be a barrier to adoption. If ATM is to become the backbone of international communications, as intended, it must be available at low cost to every user who wants it. Fourth, the new system must be able to work with and support the existing telecommunications hierarchies (local loops, local providers, long-distance carriers, and so on). Fifth, the new system must be connection-oriented to ensure accurate and predictable delivery. And last but not least, one objective is to move as many of the functions to hardware as possible (for speed) and eliminate as many software functions as possible (again for speed).

### Cell Networks

ATM is a *cell network*. A **cell** is a small data unit of fixed size that is the basic unit of data exchange in a cell network. In this type of network, all data are loaded into identical cells that can be transmitted with complete predictability and uniformity. Cells are multiplexed with other cells and routed through a cell network. Because each cell is the same size and all are small, any problems associated with multiplexing different-sized packets are avoided.

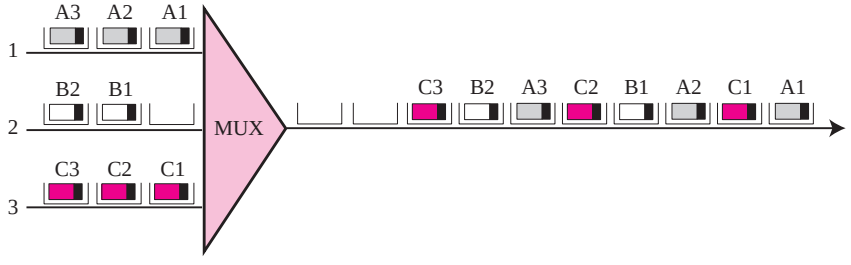
**A cell network uses the cell as the basic unit of data exchange.**
**A cell is defined as a small, fixed-size block of information.**

### Asynchronous TDM

ATM uses **asynchronous time-division multiplexing**—that is why it is called Asynchronous Transfer Mode—to multiplex cells coming from different channels. It uses fixed-size slots the size of a cell. ATM multiplexers fill a slot with a cell from any input channel that has a cell; the slot is empty if none of the channels has a cell to send.

Figure 3.32 shows how cells from three inputs are multiplexed. At the first tick of the clock, channel 2 has no cell (empty input slot), so the multiplexer fills the slot with a cell from the third channel. When all the cells from all the channels are multiplexed, the output slots are empty.

**Figure 3.32** *ATM multiplexing*

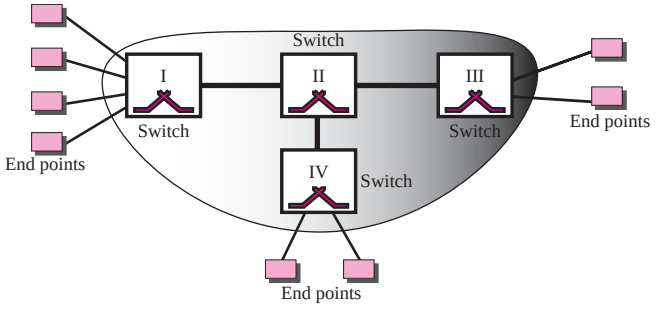



### ATM Architecture

ATM is a switched network. The user access devices, called the end points, are connected to the switches inside the network. The switches are connected to each other using high-speed communication channels. Figure 3.33 shows an example of an ATM network.

**Virtual Connection** Connection between two end points is accomplished through transmission paths (TPs), virtual paths (VPs), and virtual circuits (VCs). A **transmission path (TP)** is the physical connection (wire, cable, satellite, and so on) between an end point and a switch or between two switches. Think of two switches as two cities. A transmission path is the set of all highways that directly connects the two cities.

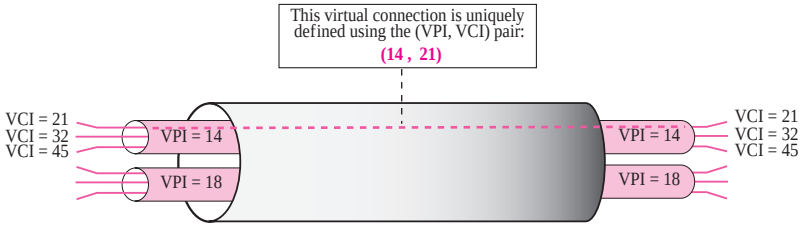
**Figure 3.33** *Architecture of an ATM network*




A transmission path is divided into several virtual paths. A **virtual path (VP)** provides a connection or a set of connections between two switches. Think of a virtual path as a highway that connects two cities. Each highway is a virtual path; the set of all highways is the transmission path.

Cell networks are based on **virtual circuits (VCs).** All cells belonging to a single message follow the same virtual circuit and remain in their original order until they reach their destination. Think of a virtual circuit as the lanes of a highway (virtual path) as shown in Figure 3.34.

**Figure 3.34** *Virtual circuits*




The figure also shows the relationship between a transmission path (a physical connection), virtual paths (a combination of virtual circuits that are bundled together because parts of their paths are the same), and virtual circuits that logically connect two points together.

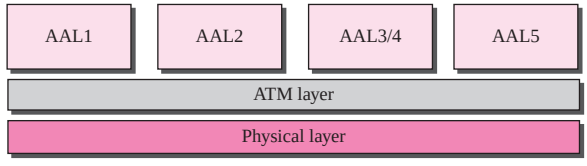
In a virtual circuit network, to route data from one end point to another, the virtual connections need to be identified. For this purpose, the designers of ATM created a hierarchical identifier with two levels: a **virtual path identifier (VPI)** and a **virtual circuit identifier (VCI).** The VPI defines the specific VP and the VCI defines a particular VC inside the VP. The VPI is the same for all virtual connections that are bundled (logically) into one VP.

**A virtual connection is defined by a pair of numbers: the VPI and the VCI.**

### *ATM Layers*

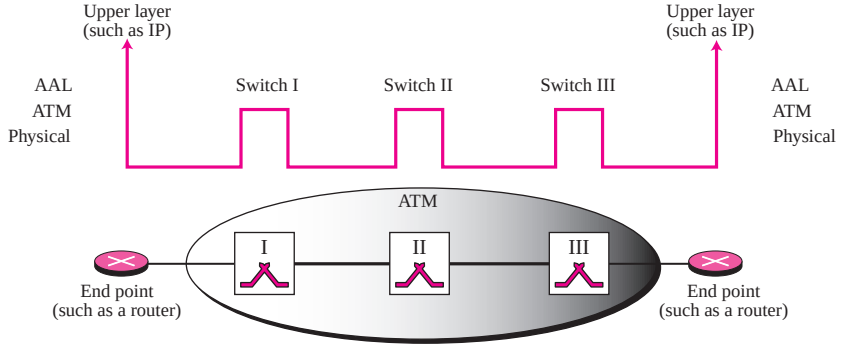
The ATM standard defines three layers. They are, from top to bottom, the application adaptation layer, the ATM layer, and the physical layer as shown in Figure 3.35.

**Figure 3.35** *ATM layers*




The physical and ATM layer are used in both switches inside the network and end points (such as routers) that use the services of the ATM. The application adaptation layer (AAL) is used only by the end points. Figure 3.36 shows the use of these layers inside and outside an ATM network.

**Figure 3.36** *Use of the layers*




### *AAL Layer*

The **application adaptation layer (AAL)** allows existing networks (such as packet networks) to connect to ATM facilities. AAL protocols accept transmissions from upper-layer services (e.g., packet data) and map them into fixed-sized ATM cells. These transmissions can be of any type (voice, data, audio, video) and can be of variable or fixed rates. At the receiver, this process is reversed—segments are reassembled into their original formats and passed to the receiving service. Although four AAL layers have been defined the one which is of interest to us is AAL5, which is used to carry IP packets in the Internet.

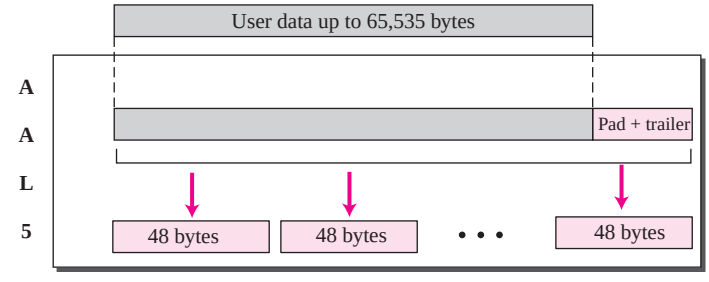
**AAL5**, which is sometimes called the **simple and efficient adaptation layer (SEAL)**, assumes that all cells belonging to a single message travel sequentially and that control functions are included in the upper layers of the sending application. AAL5

is designed for connectionless packet protocols that use a datagram approach to routing (such as the IP protocol in TCP/IP).

**The IP protocol uses the AAL5 sublayer.**

AAL5 accepts an IP packet of no more than 65,535 bytes and adds an 8-byte trailer as well as any padding required to ensure that the position of the trailer falls where the receiving equipment expects it (at the last 8 bytes of the last cell). See Figure 3.37. Once the padding and trailer are in place, AAL5 passes the message in 48-byte segments to the ATM layer.

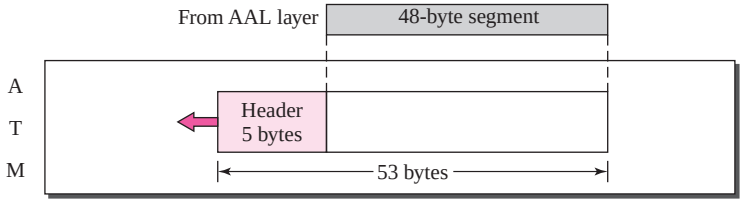
**Figure 3.37** *AAL5*




### *ATM Layer*

The ATM layer provides routing, traffic management, switching, and multiplexing services. It processes outgoing traffic by accepting 48-byte segments from the AAL sublayer. The addition of a 5-byte header transforms the segment into a 53-byte cell (see Figure 3.38).

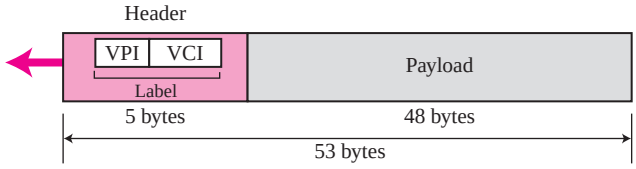
**Figure 3.38** *ATM layer*




A cell is 53 bytes in length with 5 bytes allocated to header and 48 bytes carrying payload (user data may be less than 48 bytes). Most of the header is occupied by the VPI and VCI. Figure 3.39 shows the cell structure.

The combination of VPI and VCI can be thought of as a *label* that defines a particular virtual connection.

**Figure 3.39** *An ATM cell*



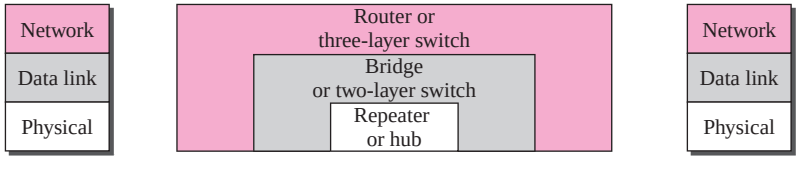

### *Physical Layer*

The physical layer defines the transmission medium, bit transmission, encoding, and electrical to optical transformation. It provides convergence with physical transport protocols, such as SONET and T-3, as well as the mechanisms for transforming the flow of cells into a flow of bits.

## 3.5 CONNECTING DEVICES

LANs or WANs do not normally operate in isolation. They are connected to one another or to the Internet. To connect LANs and WANs together we use connecting devices. Connecting devices can operate in different layers of the Internet model. We discuss three kinds of **connecting devices:** repeaters (or hubs), bridges (or two-layer switches), and routers (or three-layer switches). Repeaters and hubs operate in the first layer of the Internet model. Bridges and two-layer switches operate in the first two layers. Routers and three-layer switches operate in the first three layers. Figure 3.40 shows the layers in which each device operates.

**Figure 3.40** *Connecting devices*

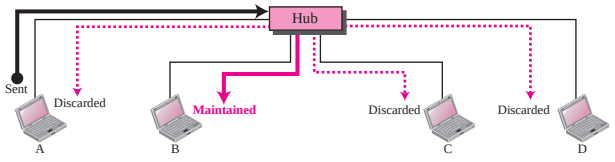



### Repeaters

A **repeater** is a device that operates only in the physical layer. Signals that carry information within a network can travel a fixed distance before attenuation endangers the integrity of the data. A repeater receives a signal and, before it becomes too weak or corrupted, *regenerates* and *retimes* the original bit pattern. The repeater then sends the refreshed signal. In the past, when Ethernet LANs were using bus topology, a repeater was used to connect two segments of a LAN to overcome the length restriction of the

coaxial cable. Today, however, Ethernet LANs use star topology. In a star topology, a repeater is a multiport device, often called a **hub,** that can be used to serve as the connecting point and at the same time function as a repeater. Figure 3.41 shows that when a packet from station A to B arrives at the hub, the signal representing the frame is regenerated to remove any possible corrupting noise, but the hub forwards the packet from all outgoing port to all stations in the LAN. In other words, the frame is broadcast. All stations in the LAN receive the frame, but only station B keeps it. The rest of the stations discard it. Figure 3.41 shows the role of a repeater or a hub in a switched LAN.

**Figure 3.41** *Repeater or hub*




The figure definitely shows that a hub does not have a filtering capability; it does not have the intelligence to find from which port the frame should be sent out.

**A repeater forwards every bit; it has no filtering capability.**

A hub or a repeater is a physical-layer device. They do not have any data-link address and they do not check the data-link address of the received frame. They just regenerate the corrupted bits and send them out from every port.

## Bridges

A **bridge** operates in both the physical and the data link layers. As a physical-layer device, it regenerates the signal it receives. As a data link layer device, the bridge can check the MAC addresses (source and destination) contained in the frame.

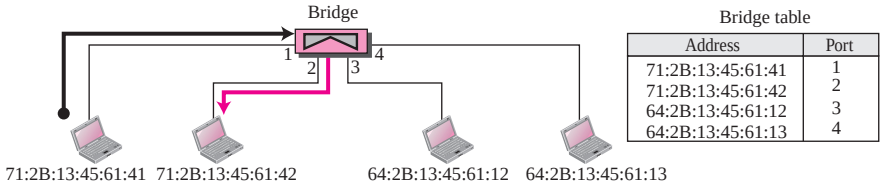
### *Filtering*

One may ask what is the difference in functionality between a bridge and a repeater. A bridge has **filtering** capability. It can check the destination address of a frame and can decide from which outgoing port the frame should be send out.

**A bridge has a table used in filtering decisions.**

Let us give an example. In Figure 3.42, we have a LAN with four stations that are connected to a bridge. If a frame destined for station 71:2B:13:45:61:42 arrives at port 1, the bridge consults its table to find the departing port. According to its table, frames for 71:2B:13:45:61:42 should be sent out only through port 2; therefore, there is no need for forwarding the frame through other ports.

**Figure 3.42** *Bridge*




**A bridge does not change the physical (MAC) addresses in a frame.**

### *Transparent Bridges*

A **transparent bridge** is a bridge in which the stations are completely unaware of the bridge's existence. If a bridge is added or deleted from the system, reconfiguration of the stations is unnecessary. According to the IEEE 802.1d specification, a system equipped with transparent bridges must meet three criteria:

1. Frames must be forwarded from one station to another.
2. The forwarding table is automatically made by learning frame movements in the network.
3. Loops in the system must be prevented.

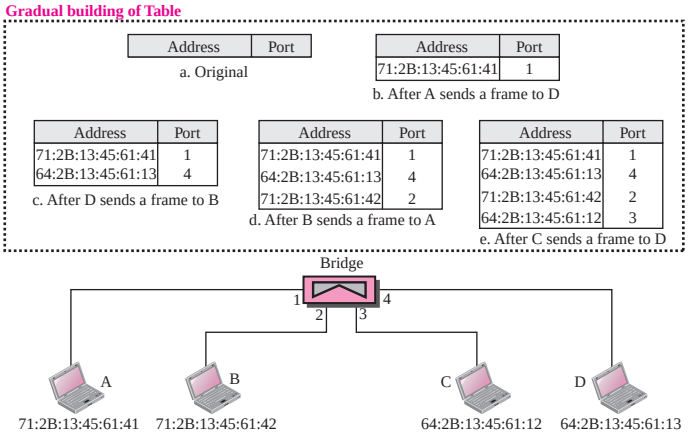
**Forwarding** A transparent bridge must correctly forward the frames, as discussed in the previous section.

**Learning** The earliest bridges had forwarding tables that were static. The system administrator would manually enter each table entry during bridge setup. Although the process was simple, it was not practical. If a station was added or deleted, the table had to be modified manually. The same was true if a station's MAC address changed, which is not a rare event. For example, putting in a new network card means a new MAC address.

A better solution to the static table is a dynamic table that maps addresses to ports automatically. To make a table dynamic, we need a bridge that gradually learns from the frame movements. To do this, the bridge inspects both the destination and the source addresses. The destination address is used for the forwarding decision (table lookup); the source address is used for adding entries to the table and for updating purposes. Let us elaborate on this process using Figure 3.43.

1. When station A sends a frame to station D, the bridge does not have an entry for either D or A. The frame goes out from all three ports; the frame floods the network. However, by looking at the source address, the bridge learns that station A must be connected to port 1. This means that frames destined for A, in the future, must be sent out through port 1. The bridge adds this entry to its table. The table has its first entry now.
2. When station D sends a frame to station B, the bridge has no entry for B, so it floods the network again. However, it adds one more entry to the table.
3. The learning process continues until the table has information about every port.

**Figure 3.43** *Learning bridge*




However, note that the learning process may take a long time. For example, if a station does not send out a frame (a rare situation), the station will never have an entry in the table.

### *Two-Layer Switch*

When we use the term *switch*, we must be careful because a switch can mean two different things. We must clarify the term by adding the level at which the device operates. We can have a two-layer switch or a three-layer switch. A **two-layer switch** performs at the physical and data link layer; it is a sophisticated bridge with faster forwarding capability.

## Routers

A **router** is a three-layer device; it operates in the physical, data link, and network layers. As a physical layer device, it regenerates the signal it receives. As a data link layer device, the router checks the physical addresses (source and destination) contained in the packet. As a network layer device, a router checks the network layer addresses (addresses in the IP layer). Note that bridges change collision domains, but routers limit broadcast domains.

**A router is a three-layer (physical, data link, and network) device.**

A router can connect LANs together; a router can connect WANs together; and a router can connect LANs and WANs together. In other words, a router is an internetworking device; it connects independent networks together to form an internetwork. According to this definition, two networks (LANs or WANs) connected by a router become an internetwork or an internet.

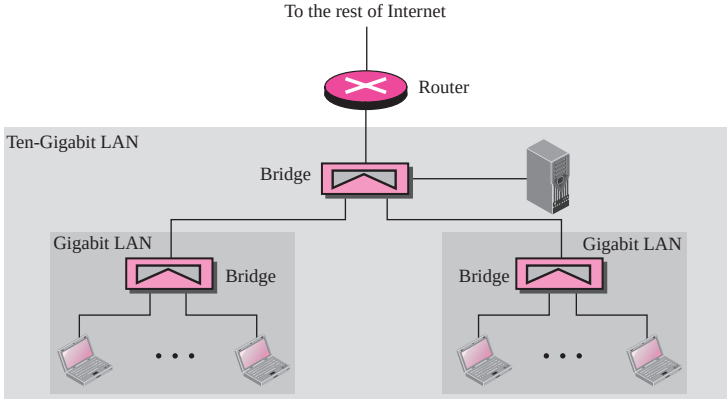
**A repeater or a bridge connects segments of a LAN.**
**A router connects independent LANs or WANs to create an internetwork (internet).**

There are three major differences between a router and a repeater or a bridge.

1. A router has a physical and logical (IP) address for each of its interfaces.
2. A router acts only on those packets in which the physical destination address matches the address of the interface at which the packet arrives.
3. A router changes the physical address of the packet (both source and destination) when it forwards the packet.

Let us give an example. In Figure 3.44, assume an organization has two separate buildings with a Gigabit Ethernet LANs installed in each building. The organization uses bridges in each LAN. The two LANs can be connected together to form a larger LAN using Ten-Gigabit Ethernet technology that speeds up the connection to the Ethernet and the connection to the organization server. A router then can connect the whole system to the Internet.

**Figure 3.44** *Routing example*




A router as we saw in Chapter 2, will change the MAC address it receives because the MAC addresses have only local jurisdictions.

We will learn more about routers and routing in future chapters after we have discussed IP addressing.

### *Three-Layer Switch*

A **three-layer switch** is a router; a router with an improved design to allow better performance. A three-layer switch can receive, process, and dispatch a packet much faster than a traditional router even though the functionality is the same. In this book, to avoid confusion, we use the term router for a three-layer switch.

**A router changes the physical addresses in a packet.**

## 3.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books: [For 07], [For 03], [Tan 03], and [Gar &Wid 04]. The items enclosed in brackets refer to the reference list at the end of the book.

## 3.7 KEY TERMS

AAL5
access point (AP)
application adaptation layer (AAL)
asymmetric digital subscriber line (ADSL)
asynchronous time-division multiplexing
Asynchronous Transfer Mode (ATM)
autonegotiation
bandwidth on demand
basic service set (BSS)
Bluetooth
bridge
BSS-transition mobility
cable modem (CM)
cable modem transmission system (CMTS)
cable TV
carrier extension
carrier sense multiple access (CSMA)
carrier sense multiple access with collision avoidance (CSMA/CA)
carrier sense multiple access with collision detection (CSMA/CD)
cell
community antenna TV (CATV)
connecting device
digital subscriber line (DSL)
digital subscriber line access multiplexer (DSLAM)
distributed interframe space (DIFS)
downloading
downstream data band
ESS-transition mobility
Ethernet
extended service set (ESS)
Fast Ethernet
fiber node
filtering
frame bursting
Frame Relay
Gigabit Ethernet
handshaking period
head end
hexadecimal notation
high bit rate digital subscriber line (HDSL)
hub
hybrid fiber-coaxial (HFC) network
IEEE 802.11
jamming signal
Link Control Protocol (LCP)
logical link control (LLC)
media access control (MAC)
network allocation vector (NAV)
Network Control Protocol (NCP)
network interface card (NIC)
no-transition mobility
optical carrier (OC)
piconet
point coordination function (PCF)
Point-to-Point Protocol (PPP)
PPP over Ethernet (PPPoE)
primary
Project 802
repeater
router
scatternet
secondaries
short interframe space (SIFS)
simple and efficient adaptation layer (SEAL)
Standard Ethernet
symmetric digital subscriber line (SDSL)
Synchronous Optical Network (SONET)
synchronous transport signal (STS)
T lines
T-1 line
T-3 line
Ten-Gigabit Ethernet
three-layer switch
transmission path (TP)
transparent bridge
two-layer switch
uploading

upstream data band
V.90
V.92
very high bit rate digital subscriber line (VDSL)
video band
virtual circuit (VC)
virtual circuit identifier (VCI)
virtual path (VP)
virtual path identifier (VPI)
wireless LAN
X.25

## 3.8 SUMMARY

- A local area network (LAN) is a computer network that is designed for a limited geographic area. The LAN market has seen several technologies such as Ethernet, token ring, token bus, FDDI, and ATM LAN. Some of these technologies survived for a while, but Ethernet is by far the dominant technology. Ethernet has gone through a long evolution. The most dominant versions of Ethernet today are Gigabit and Ten-Gigabit Ethernet.
- One of the dominant standards for wireless LAN is the one defined under IEEE 802.11 standard and sometimes called wireless Ethernet. Another popular technology is Bluetooth, which is a wireless LAN technology designed to connect devices of different functions such as telephones, notebooks, computers (desktop and laptop), cameras, printers, coffee makers, and so on.
- A point-to-point WAN technology provides a direct connection to the Internet using regular telephone lines and traditional modems, DSL lines, cable modems, T-lines, or SONET networks. The Point-to-Point Protocol (PPP) was designed for users who need a reliable point-to-point connection to the Internet. PPP operates at the physical and data link layers of the OSI model.
- A switched WAN technology provides a backbone connection in the Internet. Asynchronous Transfer Mode (ATM) is the cell relay protocol designed to support the transmission of data, voice, and video through high data-rate transmission media such as fiber-optic cable.
- Connecting devices can connect segments of a network together; they can also connect networks together to create an internet. There are three types of connecting devices: repeaters (hubs), bridges (two-layer switches), and routers (three-layer switches). Repeaters regenerate a signal at the physical layer. A hub is a multiport repeater. Bridges have access to station addresses and can forward or filter a packet in a network. They operate at the physical and data link layers. A two-layer switch is a sophisticated bridge. Routers determine the path a packet should take. They operate at the physical, data link, and network layers. A three-layer switch is a sophisticated router.

## 3.9 PRACTICE SET

### Exercises

**1.** Imagine the length of a 10Base5 cable is 2500 meters. If the speed of propagation in a thick coaxial cable is 200,000,000 meters/second, how long does it take for a

bit to travel from the beginning to the end of the network? Ignore any propagation delay in the equipment.

2. Using the data in Exercise 2, find the maximum time it takes to sense a collision. The worst case occurs when data are sent from one end of the cable and the collision happens at the other end. Remember that the signal needs to make a round trip.
3. The data rate of 10Base5 is 10 Mbps. How long does it take to create the smallest frame? Show your calculation.
4. Using the data in Exercises 3 and 4, find the minimum size of an Ethernet frame for collision detection to work properly.
5. An Ethernet MAC sublayer receives 42 bytes of data from the LLC sublayer. How many bytes of padding must be added to the data?
6. An Ethernet MAC sublayer receives 1510 bytes of data from the LLC layer. Can the data be encapsulated in one frame? If not, how many frames need to be sent? What is the size of the data in each frame?
7. Compare and contrast CSMA/CD with CSMA/CA.
8. Use Table 3.10 to compare and contrast the fields in IEEE 802.3 and 802.11.

**Table 3.10** *Exercise 8*

| *Fields* | *IEEE 802.3 Field Size* | *IEEE 802.11 Field Size* |
|---|---|---|
| Destination address | | |
| Source address | | |
| Address 1 | | |
| Address 2 | | |
| Address 3 | | |
| Address 4 | | |
| FC | | |
| D/ID | | |
| SC | | |
| PDU length | | |
| Data and padding | | |
| Frame body | | |
| FCS (CRC) | | |

## Research Activities

9. Traditional Ethernet uses a version of the CSMA/CD access method. It is called CSMA/CD with 1-persistent. Find some information about this method.
10. DSL uses a modulation technique called DMT. Find some information about this modulation technique and how it can be used in DSL.
11. PPP goes through different phases, which can be shown in a transition state diagram. Find the transition diagram for a PPP connection.
12. Find the format of an LCP packet (encapsulated in a PPP frame). Include all fields, their codes, and their purposes.

13. Find the format of an NCP packet (encapsulated in a PPP frame). Include all fields, their codes, and their purposes.
14. Find the format of an ICP packet (encapsulated in a PPP frame). Include all fields, their codes, and their purposes.
15. PPP uses two authentication protocols, PAP and CHAP. Find some information about these two protocols and how they are used in PPP.
16. Find how an IP packet can be encapsulated in ATM cells using AAL5 layer.
17. To prevent loops in a network using transparent bridges, one uses the spanning tree algorithm. Find some information about this algorithm and how it can prevent loops.

# PART 2

# Network Layer

CHAPTER

# 4

# *Introduction to Network Layer*

To solve the problem of delivery through several links, the network layer (or the internetwork layer, as it is sometimes called) was designed. The network layer is responsible for host-to-host delivery and for routing the packets through the routers. In this chapter, we give an introduction to the network layer to prepare readers for a more thorough coverage in Chapters 5 through 12. In this chapter we give the rationale for the need of the network layers and the issues involved. However, we need the next eight chapters to fully understand how these issues are answered. We may even need to cover the whole book before we get satisfactory answers for them.

## OBJECTIVES

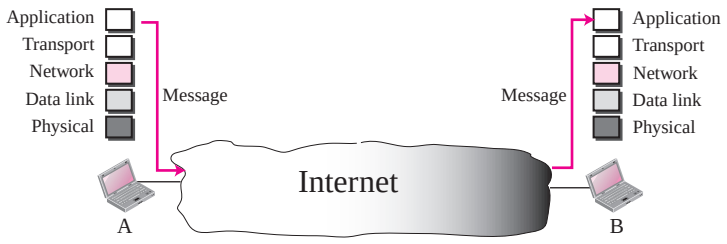
*This chapter has several objectives:*

- To introduce switching and in particular packet switching as the mechanism of data delivery in the network layer.
- To discuss two distinct types of services a packet-switch network can provide: connectionless service and connection-oriented service.
- To discuss how routers forward packets in a connectionless packet-switch network using the destination address of the packet and a routing table.
- To discuss how routers forward packets in a connection-oriented packet-switch network using the label on the packet and a routing table.
- To discuss services already provided in the network layer such as logical addressing and delivery at the source, at each router, and at the destination.
- To discuss issues or services that are not directly provided in the network layer protocol, but are sometimes provided by some auxiliary protocols or some protocols added later to the Internet.

## 4.1 INTRODUCTION

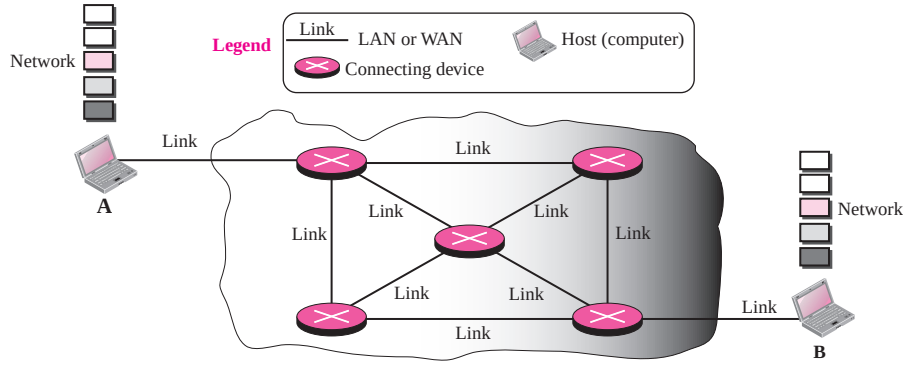
At the conceptual level, we can think of the global Internet as a black box network that connects millions (if not billions) of computers in the world together. At this level, we are only concerned that a message from the application layer in one computer reaches the application layer in another computer. In this conceptual level, we can think of communication between A and B as shown in Figure 4.1.

**Figure 4.1** *Internet as a black box*




The Internet, however, is not one single network; it is made of many networks (or links) connected together through the connecting devices. In other words, the Internet is an internetwork, a combination of LANs and WANs. To better understand the role of the network layer (or the internetwork layer), we need to move from our conceptual level and think about all of these LANs and WANs that make the Internet. Since it is impossible to show all of these LANs and WANs, we show only an imaginary small internet with a few networks and a few connecting devices, as shown in Figure 4.2.

**Figure 4.2** *Internet as a combination of LAN and WANs connected together*




An internet as a combination of links and connecting devices

In this model, a connecting device such as a router acts as a switch. When a packet arrives from one of its ports (interface), the packet is forwarded through another port to the next switch (or final destination). In other words, a process called **switching** occurs at the connecting device.

## 4.2 SWITCHING

From the previous discussion, it is clear that the passage of a message from a source to a destination involves many decisions. When a message reaches a connecting device, a decision needs to be made to select one of the output ports through which the packet needs to be send out. In other words, the connecting device acts as a switch that connects one port to another port.

### Circuit Switching

One solution to the switching is referred to as **circuit switching**, in which a physical circuit (or channel) is established between the source and destination of the message before the delivery of the message. After the circuit is established, the entire message, is transformed from the source to the destination. The source can then inform the network that the transmission is complete, which allows the network to open all switches and use the links and connecting devices for another connection. The circuit switching was never implemented at the network layer; it is mostly used at the physical layer.

**In circuit switching, the whole message is sent from the source to the destination without being divided into packets.**

#### Example 4.1

A good example of a circuit-switched network is the early telephone systems in which the path was established between a caller and a callee when the telephone number of the callee was dialed by the caller. When the callee responded to the call, the circuit was established. The voice message could now flow between the two parties, in both directions, while all of the connecting devices maintained the circuit. When the caller or callee hung up, the circuit was disconnected. The telephone network is not totally a circuit-switched network today.

### Packet Switching

The second solution to switching is called **packet switching**. The network layer in the Internet today is a packet-switched network. In this type of network, a message from the upper layer is divided into manageable packets and each packet is sent through the network. The source of the message sends the packets one by one; the destination of the message receives the packets one by one. The destination waits for all packets belonging to the same message to arrive before delivering the message to the upper layer. The connecting devices in a packet-switching network still need to decide how to route the packets to the final destination. Today, a packet-switched network can use two different

approaches to route the packets: the datagram approach and the virtual circuit approach. We discuss both approaches in the next section.

**In packet switching, the message is first divided into manageable packets at the source before being transmitted. The packets are assembled at the destination.**

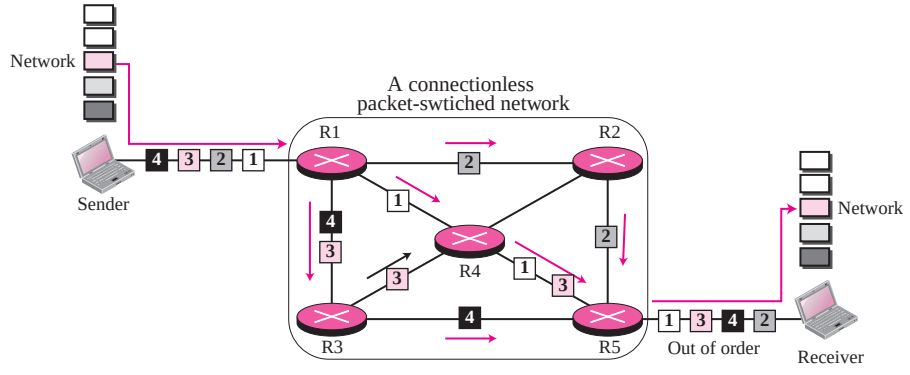
## 4.3 PACKET SWITCHING AT NETWORK LAYER

The network layer is designed as a packet-switched network. This means that the packet at the source is divided into manageable packets, normally called **datagrams**. Individual datagrams are then transferred from the source to the destination. The received datagrams are assembled at the destination before recreating the original message. The packet-switched network layer of the Internet was originally designed as a *connectionless service*, but recently there is a tendency to change this *to a connection-oriented service*. We first discuss the dominant trend and then briefly discuss the new one.

### Connectionless Service

When the Internet started, the network layer was designed to provide a **connectionless service**, in which the network layer protocol treats each packet independently, with each packet having no relationship to any other packet. The packets in a message may or may not travel the same path to their destination. When the Internet started, it was decided to make the network layer a connectionless service to make it simple. The idea was that the network layer is only responsible for delivery of packets from the source to the destination. Figure 4.3 shows the idea.

**Figure 4.3** *A connectionless packet-switched network*

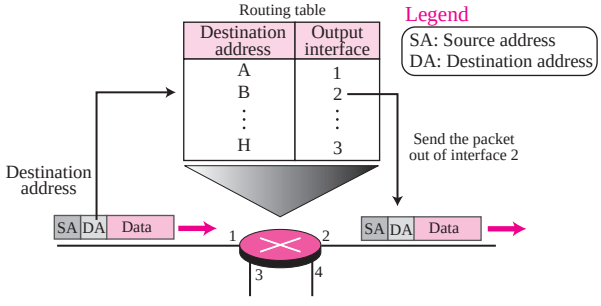



When the network layer provides a connectionless service, each packet traveling in the Internet is an independent entity; there is no relationship between packets belonging to the same message. The switches in this type of network are called *routers*. A packet

belonging to a message may be followed by a packet belonging to the same message or a different message. A packet may be followed by a packet coming from the same or from a different source.

Each packet is routed based on the information contained in its header: source and destination address. The destination address defines where it should go; the source address defines where it comes from. The router in this case routes the packet based only on the destination address. The source address may be used to send an error message to the source if the packet is discarded. Figure 4.4 shows the forwarding process in a router in this case. We have used symbolic addresses such as A and B.

**Figure 4.4** *Forwarding process in a router when used in a connectionless network*

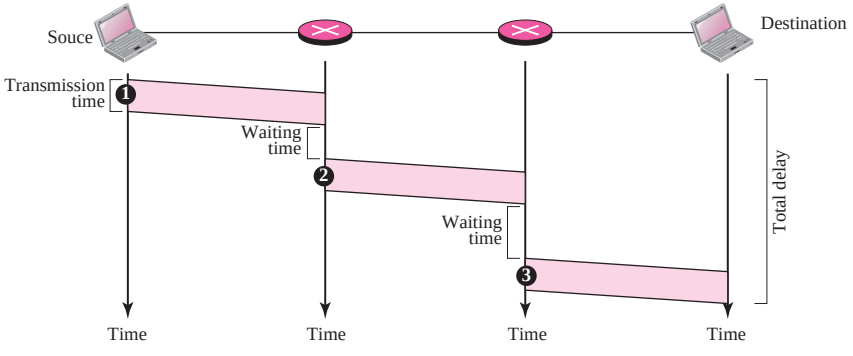



**In a connectionless packet-switched network, the forwarding decision is based on the destination address of the packet.**

### *Delay In Connectionless Network*

If we ignore the fact that the packet may be lost and resent and also the fact that the destination may be needed to wait to receive all packets, we can model the delay as shown in Figure 4.5.

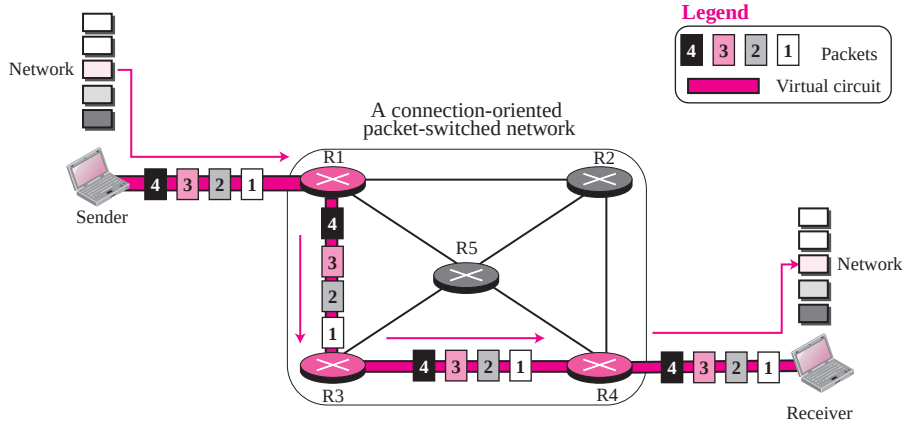
**Figure 4.5** *Delay in a connectionless network*



## Connection-Oriented Service

In a **connection-oriented service,** there is a relation between all packets belonging to a message. Before all datagrams in a message can be sent, a virtual connection should be set up to define the path for the datagrams. After connection setup, the datagrams can follow the same path. In this type of service, not only must the packet contain the source and destination addresses, it must also contain a *flow label*, a *virtual circuit identifier* that defines the virtual path the packet should follow. We will shortly show how this flow label is determined, but for the moment, we assume that the packet carries this *label*. Although it looks as though the use of the label may make the source and destination addresses useless, the parts of the Internet that use connectionless service at the network layer still keep these addresses for several reasons. One reason is that part of the packet path may still be using the connectionless service. Another reason is that the protocol at the network layer is designed with these addresses and it may take a while before they can be changed. Figure 4.6 shows the concept of connection-oriented service.

**Figure 4.6** *A connection-oriented packet switched network*



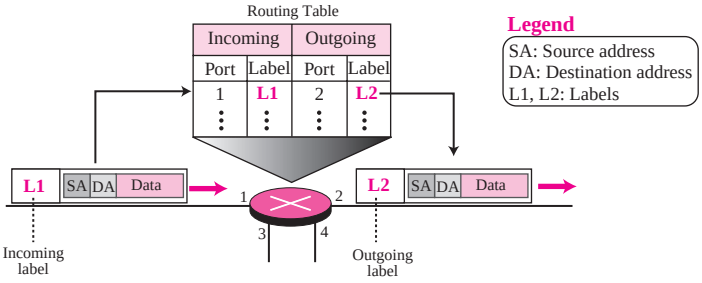

Each packet is forwarded based on the label in the packet. To follow the idea of connection-oriented design to be used in the Internet, we assume that the packet has a label when it reaches the router. Figure 4.7 shows the idea.

In this case, the forwarding decision is based on the value of the label, or virtual circuit identifier as it is sometimes called.

To create a connection-oriented service, a three-phase process is used: *setup*, *data transfer*, and *teardown*. In the setup phase, the source and destination addresses of the sender and receiver is used to make table entries for the connection-oriented service. In the teardown phase, the source and destination inform the router to delete the corresponding entries. Data transfer occurs between these two phases.

**In a connection-oriented packet switched network, the forwarding decision is based on the label of the packet.**

**Figure 4.7** *Forwarding process in a router when used in a connection-oriented network*



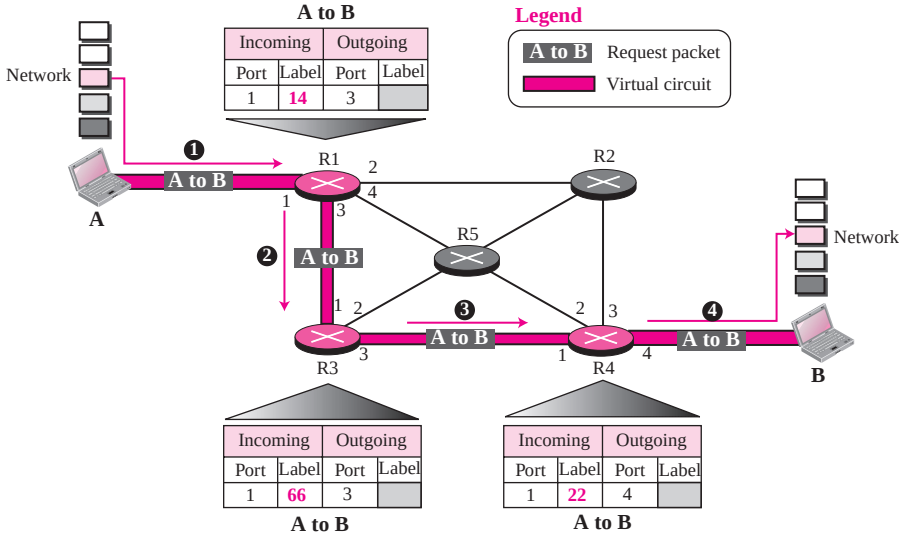

### *Setup Phase*

In the **setup phase**, a router creates an entry for a virtual circuit. For example, suppose source A needs to create a virtual circuit to destination B. Two auxiliary packets need to be exchanged between the sender and the receiver: the request packet and the acknowledgment packet.

**Request packet** A request packet is sent from the source to the destination. This auxiliary packet carries the source and destination addresses. Figure 4.8 shows the process.

1. Source A sends a request packet to router R1.
2. Router R1 receives the request packet. It knows that a packet going from A to B goes out through port 3. How the router has obtained this information is a point

**Figure 4.8** *Sending request packet in a virtual-circuit network*

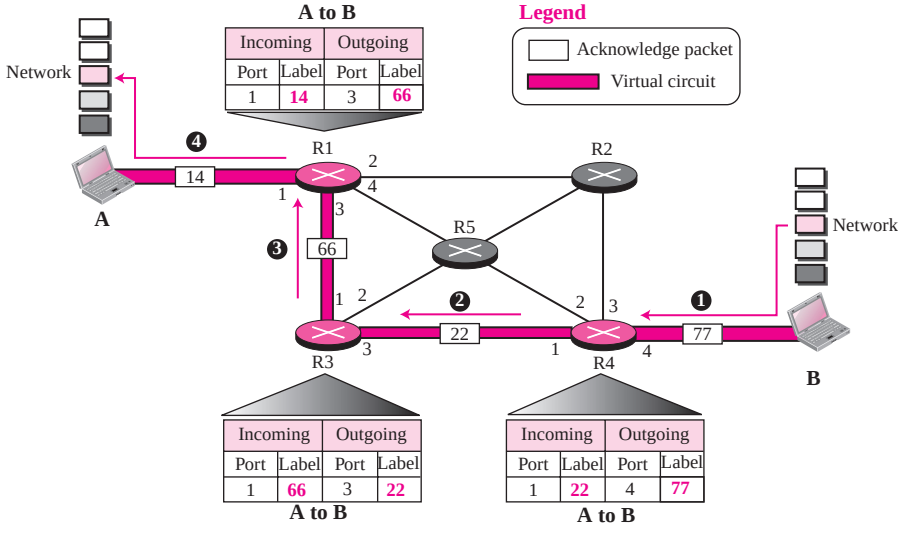


covered in future chapters. For the moment, assume that it knows the output port. The router creates an entry in its table for this virtual circuit, but it is only able to fill three of the four columns. The router assigns the incoming port (1) and chooses an available incoming label (14) and the outgoing port (3). It does not yet know the outgoing label, which will be found during the acknowledgment step. The router then forwards the packet through port 3 to router R3.

3. Router R3 receives the setup request packet. The same events happen here as at router R1; three columns of the table are completed: in this case, incoming port (1), incoming label (66), and outgoing port (2).
4. Router R4 receives the setup request packet. Again, three columns are completed: incoming port (2), incoming label (22), and outgoing port (3).
5. Destination B receives the setup packet, and if it is ready to receive packets from A, it assigns a label to the incoming packets that come from A, in this case 77. This label lets the destination know that the packets come from A, and not other sources.

**Acknowledgment Packet** A special packet, called the acknowledgment packet, completes the entries in the switching tables. Figure 4.9 shows the process.

**Figure 4.9** *Setup acknowledgment in a virtual-circuit network*



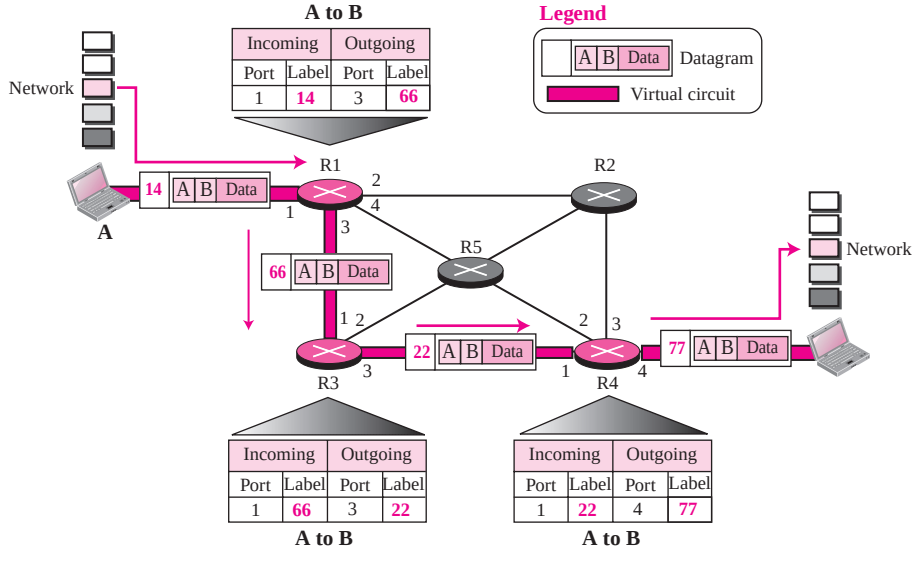

1. The destination sends an acknowledgment to router R4. The acknowledgment carries the global source and destination addresses so the router knows which entry in the table is to be completed. The packet also carries label 77, chosen by the destination as the incoming label for packets from A. Router R4 uses this label to complete the outgoing label column for this entry. Note that 77 is the incoming label for destination B, but the outgoing label for router R4.

2. Router R4 sends an acknowledgment to router R3 that contains its incoming label in the table, chosen in the setup phase. Router R3 uses this as the outgoing label in the table.
3. Router R3 sends an acknowledgment to router R1 that contains its incoming label in the table, chosen in the setup phase. Router R1 uses this as the outgoing label in the table.
4. Finally router R1 sends an acknowledgment to source A that contains its incoming label in the table, chosen in the setup phase.
5. The source uses this as the outgoing label for the data packets to be sent to destination B.

### *Data Transfer Phase*

The second phase is called the **data transfer phase**. After all routers have created their routing table for a specific virtual circuit, then the network-layer packets belonging to one message can be sent one after another. In Figure 4.10, we show the flow of one single packet, but the process is the same for 1, 2, or 100 packets. The source computer uses the label 14, which it has received from router R1 in the setup phase. Router R1 forwards the packet to router R3, but changes the label to 66. Router R3 forwards the packet to router R4, but changes the label to 22. Finally, router R4 delivers the packet to its final destination with the label 77. All the packets in the message follow the same sequence of labels to reach their destination. The packet arrives in order at the destination.

**Figure 4.10** *Flow of one packet in an established virtual circuit.*



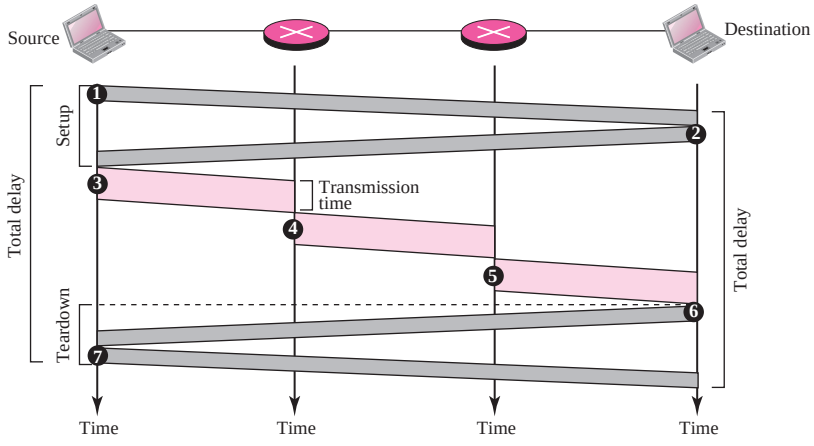

### *Teardown Phase*

In the **teardown phase,** source A, after sending all packets to B, sends a special packet called a *teardown packet*. Destination B responds with a *confirmation* packet. All routers delete the corresponding entry from their tables.

### *Delay In Connection-Oriented Network*

If we ignore the fact that the packet may be lost and resent, we can model the delay as shown in Figure 4.11.

**Figure 4.11** *Delay in a connection-oriented network*



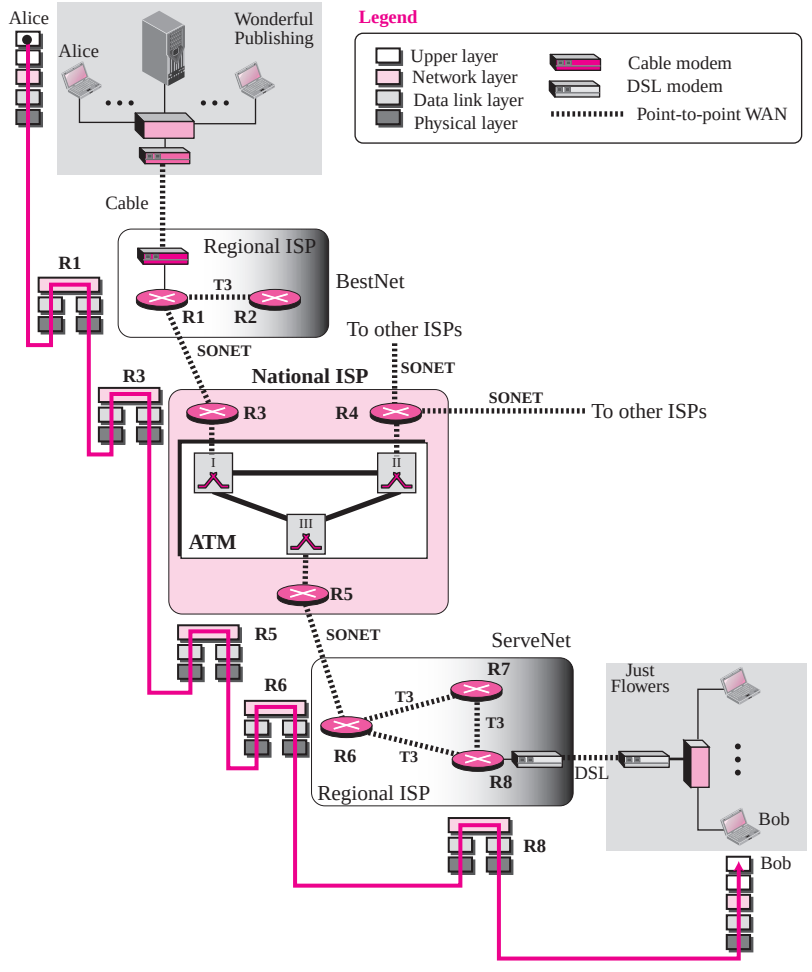

## 4.4 NETWORK LAYER SERVICES

In this section, we briefly discuss services provided by the network layer. Our discussion is mostly based on the connectionless service, the dominant service in today's Internet.

### An Example

To better understand the issues to be discussed, we give an example. In Figure 4.12, we assume Alice, who is working in a publishing company, Wonderful Publishing, needs to send a message to Bob, the manager of a flower shop, Just Flowers, to inform him that the advertising brochure for the shop has been printed and is ready to be shipped. Alice sends the message using an e-mail. Let us follow the imaginary path Alice's message takes to reach Bob. The Wonderful Publishing company uses a LAN, which is connected via a cable WAN to a regional ISP called BestNet; the Just Flowers company also uses a LAN, which is connected via a DSL WAN to another regional ISP called ServeNet. The two regional ISPs are connected through high-speed SONET WANs to a national ISP. The message that Alice sends to Bob may be split into several network-layer packets. As we will see shortly, packets may or may not follow the same path. For the sake of discussion, we follow the path of one single packet from Alice's computer to Bob's computer. We also assume that the packet passes through routers R1, R3, R5, R6, and R8 before reaching its destination. The two computers are involved in five layers; the routers are involved in three layers of the TCP/IP protocol suite.

**Figure 4.12** *An imaginary part of the Internet*



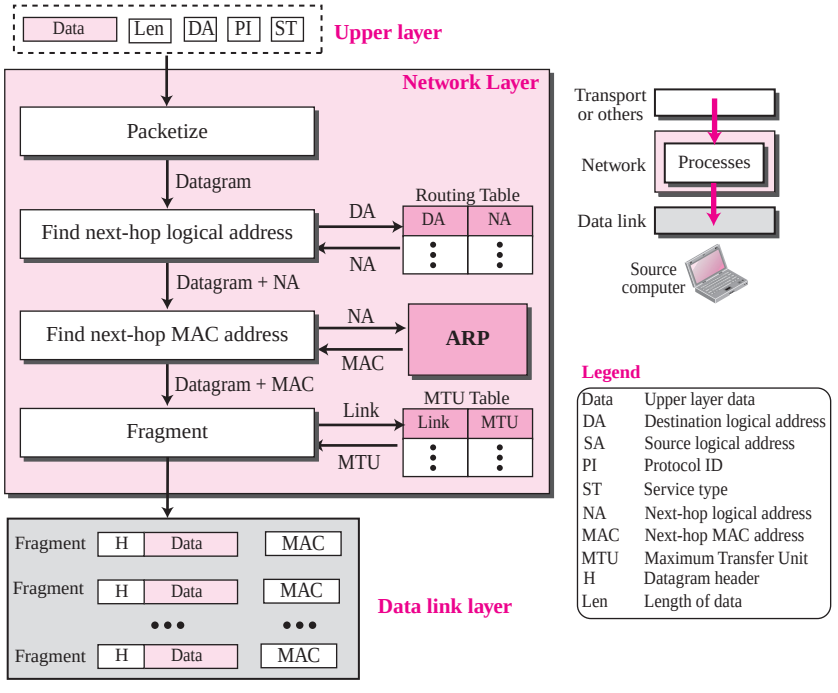

## Logical Addressing

Since the network layer provides end-to-end communication, the two computers that need to communicate with each other each need a universal identification system, referred to as network-layer address or logical address. This type of identification is provided in the network layer through a uniform and global addressing mechanism. The Internet uses an address space. Each entity that needs to use the Internet needs to be assigned a unique address from this pool. In Chapter 5 we discuss this addressing space in version 4 of the Internet; In Chapter 26, we discuss the new addressing system in version 6 (version 5 was never implemented). In Figure 4.12, Alice and Bob need two network-layer addresses to be able to communicate.

## Services Provided at the Source Computer

The network layer at the source computer provides four services: packetizing, finding the logical address of the next hop, finding the physical (MAC) address of the next hop, and fragmenting the datagram if necessary. Figure 4.13 shows these services.

**Figure 4.13** *Services provided at the source computer*




The network layer receives several pieces of information from the upper layer: data, length of data, logical destination address, protocol ID (the identifier of the protocol using the network layer), and service type (discussed later). The network layer processes these pieces of information to create a set of fragmented datagrams and the next-hop MAC address and delivered them to the data link layer. We briefly discuss each service here, but the future chapters in this part of the book explain more.

### *Packetizing*

The first duty of the network layer is to encapsulate the data coming from the upper layer in a datagram. This is done by adding a header to the data that contains the logical source and destination address of the packet, information about fragmentation, the protocol ID of the protocol that has requested the service, the data length, and possibly some options. The network layer also includes a checksum that is calculated only over the datagram header. We discuss the format of the datagram and the checksum calculation in Chapter 7. Note that the upper layer protocol only provides the logical destination

address; the logical source address comes from the network layer itself (any host needs to know its own logical address).

### *Finding Logical Address of Next Hop*

The prepared datagram contains the source and destination addresses of the packet. The datagram, as we saw before, may have to pass through many networks to reach its final destination. If the destination computer is not connected to the same network as the source, the datagram should be delivered to the next router. The source and destination address in the datagram does not tell anything about the logical address of the next hop. The network layer at the source computer needs to consult a routing table to find the logical address of the next hop.

### *Finding MAC Address of Next Hop*

The network layer does not actually deliver the datagram to the next hop; it is the duty of the data link layer to do the delivery. The data link layer needs the MAC address of the next hop to do the delivery. To find the MAC address of the next hop, the network layer could use another table to map the next-hop logical address to the MAC address. However, for the reason we discuss in Chapter 8, this task has been assigned to another auxiliary protocol called Address Resolution Protocol (ARP) that finds the MAC address of the next hop given the logical address.

### *Fragmentation*

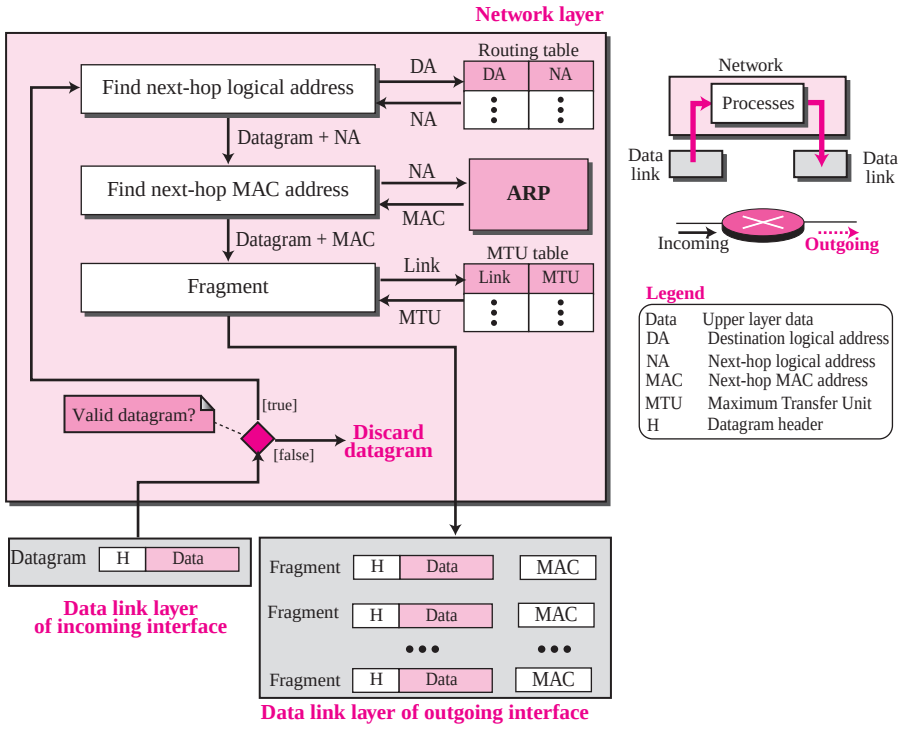
The datagram at this step may not be ready to be passed to the data link layer. As we saw in Chapter 3, most LANs and WANs have a limit on the size of the data to be carried in a frame (MTU). The datagram prepared at the network layer, may be larger than that limit. The datagram needs to be fragmented to smaller units before being passed to the data link layer. Fragmentation needs to preserve the information at the header of the datagram. In other words, although the data can be fragmented, the header needs to be repeated. In addition, some more information needs to be added to the header to define the position of the fragment in the whole datagram. We discuss fragmentation in more detail in Chapter 7.

## Services Provided at Each Router

As we have mentioned before, a router is involved with two interfaces with respect to a single datagram: the incoming interface and the outgoing interface. The network layer at the router, therefore, needs to interact with two data link layers: the data link of the incoming interface and the data link layer of the outgoing interface. The network layer is responsible to receive a datagram from the data link layer of the incoming interface, fragment it if necessary, and deliver the fragments to the data link of the outgoing interface. The router normally does not involve upper layers (with some exceptions discussed in future chapters). Figure 4.14 shows the services provides by a router at the network layer.

The three processes (finding next-hop logical address, finding next-hop MAC address, and fragmentation) here are the same as the last three processes mentioned for

**Figure 4.14** *Processing at each router*

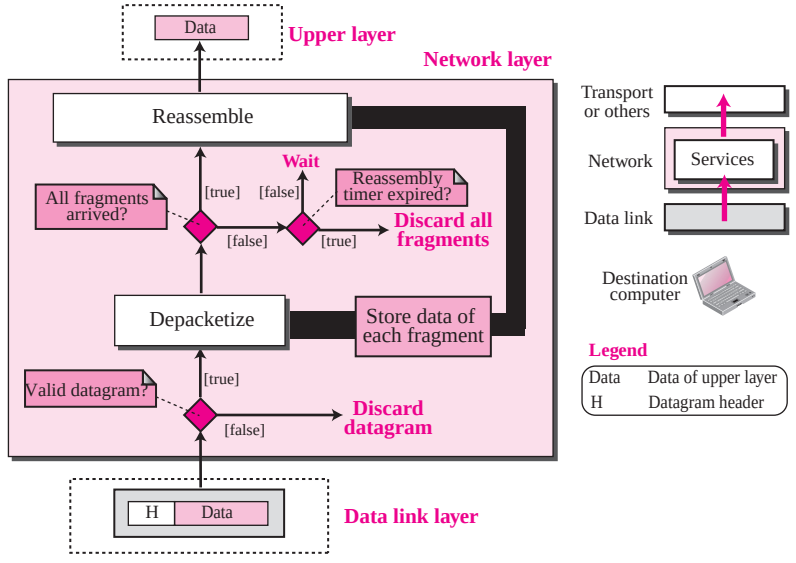



a source. Before applying these processes, however, the router needs to check the validity of the datagram using the checksum (see Chapter 7). Validation here means that the datagram header is not corrupted and the datagram is delivered to the correct router.

## Services Provided at the Destination Computer

The network layer at the destination computer is simpler. No forwarding is needed. However, the destination computer needs to assemble the fragments before delivering the data to the destination. After validating each datagram, the data is extracted from each fragment and stored. When all fragments have arrived, the data are reassembled and delivered to the upper layer. The network layer also sets a reassembly timer. If the timer is expired, all data fragments are destroyed and an error message is sent that all the fragmented datagram need to be resent. Figure 4.15 shows the process. Note that the process of fragmentation is transparent to the upper layer because the network layer does not deliver any piece of data to the upper layer until all pieces have arrived and assembled. Since a datagram may have been fragmented in the source computer as well as in any router (multilevel) fragmentation, the reassembly procedure is very delicate and complicated. We discuss the procedure in more detail in Chapter 7.

**Figure 4.15** *Processing at the destination computer*




## 4.5 OTHER NETWORK LAYER ISSUES

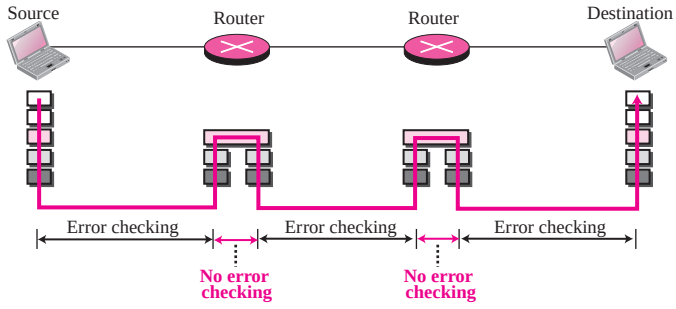
In this section we introduce some issues related to the network layer. These issues actually represent services that are normally discussed for the network layer, but they are either partially implemented at the network layer or not implemented at all. Some services are provided by some auxiliary protocols or by protocols added to the Internet later. Most of these issues resurface in future chapters.

### Error Control

**Error control** means including a mechanism for detecting corrupted, lost, or duplicate datagrams. Error control also includes a mechanism for correcting errors after they have been detected. The network layer in the Internet does not provide a real error control mechanism. At the surface level, it looks as though there is no need for error control at the network layer because each datagram passes through several networks before reaching its final destination. The data link layer that controls the behavior of these networks (LANs or WANs) use error control. In other words, if a hop-to-hop error control is already implemented at the data link layer, why do we need error control at the network layer? Although hop-to-hop error control may protect a datagram to some extent, it does not provide full protection. Figure 4.16 shows that there are some areas in the path of the datagram that some errors may occur, but never checked; the error control at the data link layer can miss any error that occurs when the datagram is being processed by the router.

The designers of the network layer wanted to make this layer operate simply and fast. They thought if there is a need for more rigorous error checking, it can be done at

**Figure 4.16** *Error checking at the data link layer*




the upper layer protocol that uses the service of the network layer. Another rationale for omitting the error checking at this layer can be related to fragmentation. Since the data is possibly fragmented at some routers and part of the network layer may be changed because of fragmentation, if we use error control, it must be checked at each router. This makes error checking at this layer very inefficient.

The designers of the network layer, however, have added a checksum field (see Chapter 7) to the datagram to control any corruption in the header, but not the whole datagram. This checksum may prevent any changes or corruptions in the header of the datagram between two hops and from end to end. For example, it prevents the delivery of the datagram to a wrong destination if the destination address has been corrupted. However, since the header may be changed in each router, the checksum needs to be calculated at the source and recalculated at each router.

We need to mention that although the network layer at the Internet does not directly provide error control, the Internet uses another protocol, ICMP, that provides some kind of error control if the datagram is discarded or has some unknown information in the header. We discuss ICMP in detail in Chapter 9.

## Flow Control

**Flow control** regulates the amount of data a source can send without overwhelming the receiver. If the upper layer at the source computer produces data faster than the upper layer at the destination computer can consume it, the receiver will be overwhelmed with data. To control the flow of data, the receiver needs to send some feedback to the sender to inform the latter it is overwhelmed with data.

The network layer in the Internet, however, does not directly provide any flow control. The datagrams are sent by the sender when they are ready without any attention to the readiness of the receiver.

**No flow control is provided for the current version of Internet network layer.**

Probably a few reasons for the lack of flow control in the design of the network layer can be mentioned. First, since there is no error control in this layer, the job of the

network layer at the receiver is so simple that it may rarely be overwhelmed. Second, the upper layers that use the service of the network layer can implement buffers to receive data from the network layer as soon as they are ready and does not have to consume the data as fast as received. Second, the flow control is provided for most of the upper layer protocols that use the services of the network layer, so another level of flow control makes the network layer more complicated and the whole system less proficient.

## Congestion Control

Another issue in a network layer protocol is **congestion control.** *Congestion* in the network layer is a situation in which too many datagrams are present in an area of the Internet. Congestion may occur if the number of datagrams sent by source computers are beyond the capacity of the network or routers. In this situation, some routers may drop some of the datagrams. However, as more datagrams are dropped, the situation may become worse because, due to the error control mechanism at the upper layers, the sender may send duplicates of the lost packets. If the congestion continues, sometimes a situation may reach a point that collapses the system and no datagram is delivered.

### *Congestion Control in a Connectionless Network*

There are several ways to control congestion in a connectionless network. One solution is referred to as signaling. In backward signaling a bit can set in the datagram moving in the direction opposite to the congested direction to inform the sender that congestion has occurred and the sender needs to slow down the sending of packets. In this case, the bit can be set in the response to a packet or in a packet that acknowledges the packet. If no feedback (acknowledgment) is used at the network layer, but the upper layer uses feedback, forward signaling can be used in which a bit is set in the packet traveling in the direction of the congestion to warn the receiver of the packet about congestion. The receiver then may inform the upper layer protocol, which in turn may inform the source. No forward or backward signaling is used in the Internet network layer.

Congestion in a connectionless network can also be implemented using a **choke packet,** a special packet that can be sent from a router to the sender when it encounters congestion. This mechanism, in fact, is implemented in the Internet network layer. The network layer uses an auxiliary protocol, ICMP, which we discuss in Chapter 9. When a router is congested, it can send an ICMP packet to the source to slow down.

Another way to ameliorate the congestion is to rank the packets by their *importance* in the whole message. A field can be used in the header of a packet to define the rank of a datagram as more important or less important, for example. If a router is congested and needs to drop some packets, the packets marked as less important can be dropped. For example, if a message represents an image, it may be divided into many packets. Some packets, the one in the corner, may be less important than the ones representing the middle part of the image. If the router is congested, it can drop these less important packets without tremendously changing the quality of the image. We talk about these issues in Chapter 25 when we discuss multimedia communication.

### *Congestion Control in a Connection-Oriented Network*

It is sometimes easier to control congestion in a connection-oriented network than in a connectionless network. One method simply creates an extra virtual circuit when there

is a congestion in an area. This, however, may create more problems for some routers. A better solution is *advanced negotiation* during the setup phase. The sender and the receiver may agree to a level of traffic when they setup the virtual circuit. The traffic level can be dictated by the routers that allow the establishment of the virtual circuits. In other words, a router can look at the exiting traffic and compare it with its maximum capacity, which allows a new virtual circuit to be created.

### Quality of Service

As the Internet has allowed new applications such as multimedia communication (in particular real-time communication of audio and video), the **quality of service (QoS)** of the communication has become more and more important. The Internet has thrived to provide better quality of service to support these applications. However, to keep the network layer untouched, these provisions are mostly implemented in the upper layer. Since QoS manifests itself more when we use multimedia communication, we discuss this issue in Chapter 25 when we discuss multimedia.

### Routing

A very important issue in the network layer is **routing;** how a router creates its routing table to help in forwarding a datagram in a connectionless service or helps in creating a virtual circuit, during setup phase, in a connection-oriented service. This can be done by *routing protocols*, that help hosts and routers make their routing table, maintain them, and update them. These are separate protocols that sometimes use the service of the network layer and sometimes the service of some transport layer protocols to help the network layer do its job. They can be grouped into two separate categories: unicast and multicast. We devote Chapter 11 to unicast routing and Chapter 12 to multicast routing. We need to assume that the routers already have created their routing protocols until we discuss these protocols in Chapters 11 and 12.

### Security

Another issue related to the communication at the network layer is security. Security was not a concern when the Internet was originally designed because it was used by a small number of users at the universities to do research activities; other people had no access to the Internet. The network layer was designed with no security provision. Today, however, security is a big concern. To provide security for a connectionless network layer, we need to have another virtual level that changes the connectionless service to a connection-oriented service. This virtual layer, called IPSec, is discussed in Chapter 30 after we discuss the general principles of cryptography and security in Chapter 29.

## 4.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books: [Ste 94], [Tan 03], [Com 06], [Gar & Vid 04], and [Kur & Ros 08]. The items enclosed in brackets refer to the reference list at the end of the book.

## 4.7 KEY TERMS

choke packet
circuit switching
congestion control
connectionless service
connection-oriented service
data transfer phase
error control
flow control
packet switching
quality of service (QoS)
routing
setup phase
switching
teardown phase

## 4.8 SUMMARY

- At the conceptual level, we can think of the global Internet as a black box network. The Internet, however, is not one single network; it is made of many networks (or links) connected together through the connecting devices.
- In this Internet, a connecting device such as a router acts as a switch. Two types of switching are traditionally used in networking: circuit switching and packet switching.
- The network layer is designed as a packet-switched network. Packet-switched network can provide either a connectionless service or a connection-oriented service. When the network layer provides a connectionless service, each packet traveling in the Internet is an independent entity; there is no relationship between packets belonging to the same message. In a connection-oriented service, there is a virtual connection between all packets belonging to a message.
- In a connectionless service, the packets are forwarded to the next hop using the destination address in the packet. In a connection-oriented service, the packets are forwarded to the next hop using a label in the packet.
- In a connection-oriented network, communication occurs in three phases: setup, data transfer, and teardown. After connection setup, a virtual circuit is established between the sender and the receiver in which all packets belonging to the same message are sent through that circuit.
- We discussed existing services at the network layer in the Internet including addressing, services provided at the source computer, services provided at the destination computer, and services provided at each router.
- We also discussed some issues related to the network layer, services that are normally discussed for the network layer, but they are either partially implemented at the network layer or not implemented at all. Some of these services, such as routing and security are provided by other protocols in the Internet.

## 4.9 PRACTICE SET

### Exercises

1. Give some advantages and disadvantages of the connectionless service.
2. Give some advantages and disadvantages of the connection-oriented service.

3. If a label in a connection-oriented service is *n* bits, how many virtual circuits can be established at the same time?
4. Assume a destination computer receives messages from several computers. How can it be sure that the fragments from one source is not mixed with the fragments from another source.
5. Assume a destination computer receives several packets from a source. How can it be sure that the fragments belonging to a datagram are not mixed from the fragments belonging to another datagram.
6. Why do you think that the packets in Figure 4.7 need both addresses and labels?
7. Compare and contrast the delays in connectionless and connection-oriented services. Which service creates less delay if the message is large? Which service creates less delay if the message is small?
8. In Figure 4.13, why should the fragmentation be the last service?
9. Discuss why we need fragmentation at each router.
10. Discuss why we need to do reassembly at the final destination, not at each router.
11. In Figure 4.15, why do we need to set a timer and destroy all fragments if the timer expires? What criteria do you use in selecting the expiration duration of such a timer?

CHAPTER

5

# IPv4 Addresses

At the network layer, we need to uniquely identify each device on the Internet to allow global communication between all devices. In this chapter, we discuss the addressing mechanism related to the prevalent IPv4 protocol called IPv4 addressing. It is believed that IPv6 protocol will eventually supersede the current protocol, and we need to become aware of IPv6 addressing as well. We discuss IPv6 protocol and its addressing mechanism in Chapters 26 to 28.

## OBJECTIVES

*This chapter has several objectives:*

- To introduce the concept of an address space in general and the address space of IPv4 in particular.
- To discuss the classful architecture, classes in this model, and the blocks of addresses available in each class.
- To discuss the idea of hierarchical addressing and how it has been implemented in classful addressing.
- To explain subnetting and supernetting for classful architecture and show how they were used to overcome the deficiency of classful addressing.
- To discuss the new architecture, classless addressing, that has been devised to solve the problems in classful addressing such as address depletion.
- To show how some ideas borrowed from classful addressing such as subnetting can be easily implemented in classless addressing.
- To discuss some special blocks and some special addresses in each block.
- To discuss NAT technology and show how it can be used to alleviate the shortage in number of addresses in IPv4.

## 5.1 INTRODUCTION

The identifier used in the IP layer of the TCP/IP protocol suite to identify each device connected to the Internet is called the Internet address or **IP address.** An IPv4 address is a 32-bit address that *uniquely* and *universally* defines the connection of a host or a router to the Internet; an IP address is the address of the interface.

**An IPv4 address is 32 bits long.**

IPv4 addresses are *unique*. They are unique in the sense that each address defines one, and only one, connection to the Internet. Two devices on the Internet can never have the same address at the same time. However, if a device has two connections to the Internet, via two networks, it has two IPv4 addresses. The IPv4 addresses are *universal* in the sense that the addressing system must be accepted by any host that wants to be connected to the Internet.

**The IPv4 addresses are unique and universal.**

### Address Space

A protocol like IPv4 that defines addresses has an **address space.** An address space is the total number of addresses used by the protocol. If a protocol uses *b* bits to define an address, the address space is $2^b$ because each bit can have two different values (0 or 1). IPv4 uses 32-bit addresses, which means that the address space is $2^{32}$ or 4,294,967,296 (more than four billion). Theoretically, if there were no restrictions, more than 4 billion devices could be connected to the Internet.

**The address space of IPv4 is $2^{32}$ or 4,294,967,296.**

### Notation

There are three common notations to show an IPv4 address: binary notation (base 2), dotted-decimal notation (base 256), and hexadecimal notation (base 16). The most prevalent, however, is base 256. These bases are defined in Appendix B. We also show how to convert a number from one base to another in that appendix. We recommend a review of this appendix before continuing with this chapter.

**Numbers in base 2, 16, and 256 are discussed in Appendix B.**

### *Binary Notation: Base 2*

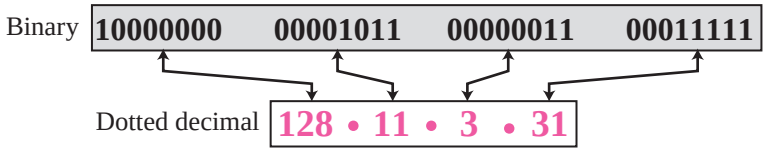
In **binary notation,** an IPv4 address is displayed as 32 bits. To make the address more readable, one or more spaces is usually inserted between each octet (8 bits). Each octet is often referred to as a byte. So it is common to hear an IPv4 address referred to as a 32-bit address, a 4-octet address, or a 4-byte address. The following is an example of an IPv4 address in binary notation:

```
01110101 10010101 00011101 11101010
```

### *Dotted-Decimal Notation: Base 256*

To make the IPv4 address more compact and easier to read, an IPv4 address is usually written in decimal form with a decimal point (dot) separating the bytes. This format is referred to as **dotted-decimal notation.** Figure 5.1 shows an IPv4 address in dotted-decimal notation. Note that because each byte (octet) is only 8 bits, each number in the dotted-decimal notation is between 0 and 255.

**Figure 5.1** *Dotted-decimal notation*




### Example 5.1

Change the following IPv4 addresses from binary notation to dotted-decimal notation.

**a.** `10000001 00001011 00001011 11101111`
**b.** `11000001 10000011 00011011 11111111`
**c.** `11100111 11011011 10001011 01101111`
**d.** `11111001 10011011 11111011 00001111`

**Solution**

We replace each group of 8 bits with its equivalent decimal number (see Appendix B) and add dots for separation:

**a.** 129.11.11.239
**b.** 193.131.27.255
**c.** 231.219.139.111
**d.** 249.155.251.15

### Example 5.2

Change the following IPv4 addresses from dotted-decimal notation to binary notation.

**a.** 111.56.45.78
**b.** 221.34.7.82

**c.** 241.8.56.12
**d.** 75.45.34.78

**Solution**
We replace each decimal number with its binary equivalent (see Appendix B):

**a.** **01101111 00111000 00101101 01001110**
**b.** **11011101 00100010 00000111 01010010**
**c.** **11110001 00001000 00111000 00001100**
**d.** **01001011 00101101 00100010 01001110**

### Example 5.3

Find the error, if any, in the following IPv4 addresses:

**a.** 111.56.045.78
**b.** 221.34.7.8.20
**c.** 75.45.301.14
**d.** 11100010.23.14.67

**Solution**

**a.** There should be no leading zeroes in dotted-decimal notation (045).
**b.** We may not have more than 4 bytes in an IPv4 address.
**c.** Each byte should be less than or equal to 255; 301 is outside this range.
**d.** A mixture of binary notation and dotted-decimal notation is not allowed.

### *Hexadecimal Notation: Base 16*

We sometimes see an IPv4 address in **hexadecimal notation.** Each hexadecimal digit is equivalent to four bits. This means that a 32-bit address has 8 hexadecimal digits. This notation is often used in network programming.

### Example 5.4

Change the following IPv4 addresses from binary notation to hexadecimal notation.

**a.** **10000001 00001011 00001011 11101111**
**b.** **11000001 10000011 00011011 11111111**

**Solution**
We replace each group of 4 bits with its hexadecimal equivalent (see Appendix B). Note that hexadecimal notation normally has no added spaces or dots; however, 0X (or 0x) is added at the beginning or the subscript 16 at the end to show that the number is in hexadecimal.

**a.** 0X810B0BEF or $810B0BEF_{16}$
**b.** 0XC1831BFF or $C1831BFF_{16}$

## Range of Addresses

We often need to deal with a range of addresses instead of one single address. We sometimes need to find the number of addresses in a range if the first and last address is given. Other times, we need to find the last address if the first address and the number of addresses in the range are given. In this case, we can perform subtraction or addition

operations in the corresponding base (2, 256, or 16). Alternatively, we can covert the addresses to decimal values (base 10) and perform operations in this base.

### Example 5.5

Find the number of addresses in a range if the first address is 146.102.29.0 and the last address is 146.102.32.255.

**Solution**

We can subtract the first address from the last address in base 256 (see Appendix B). The result is 0.0.3.255 in this base. To find the number of addresses in the range (in decimal), we convert this number to base 10 and add 1 to the result.

$$\textbf{Number of addresses} = (0 \times 256^3 + 0 \times 256^2 + 3 \times 256^1 + 255 \times 256^0) + 1 = 1024$$

### Example 5.6

The first address in a range of addresses is 14.11.45.96. If the number of addresses in the range is 32, what is the last address?

**Solution**

We convert the number of addresses minus 1 to base 256, which is 0.0.0.31. We then add it to the first address to get the last address. Addition is in base 256.

$$\textbf{Last address} = (14.11.45.96 + 0.0.0.31)_{256} = 14.11.45.127$$

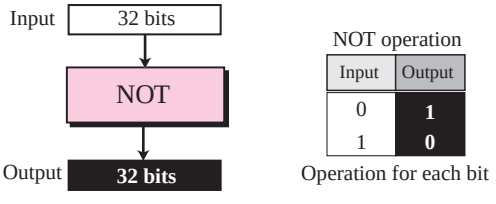
## Operations

We often need to apply some operations on 32-bit numbers in binary or dotted-decimal notation. These numbers either represent IPv4 addresses or some entities related to IPv4 addresses (such as a *mask*, which is discussed later). In this section, we introduce three operations that are used later in the chapter: NOT, AND, and OR.

### *Bitwise NOT Operation*

The bitwise NOT operation is a unary operation; it takes one input. When we apply the NOT operation on a number, it is often said that the number is complemented. The NOT operation, when applied to a 32-bit number in binary format, inverts each bit. Every 0 bit is changed to a 1 bit; every 1 bit is changed to a 0 bit. Figure 5.2 shows the NOT operation.

**Figure 5.2** *Bitwise NOT operation*



Although we can directly use the NOT operation on a 32-bit number, when the number is represented as a four-byte dotted-decimal notation, we can use a short cut; we can subtract each byte from 255.

### Example 5.7

The following shows how we can apply the NOT operation on a 32-bit number in binary.

| | | | | |
|---|---|---|---|---|
| **Original number:** | 00010001 | 01111001 | 00001110 | 00100011 |
| **Complement:** | 11101110 | 10000110 | 11110001 | 11011100 |

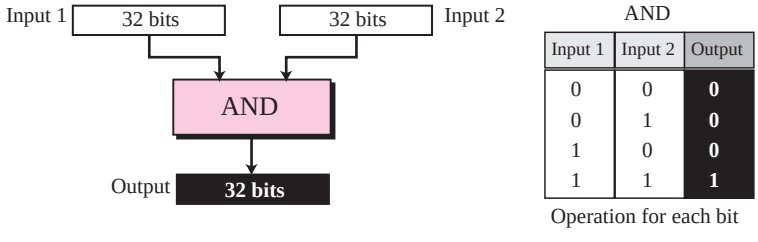
We can use the same operation using the dotted-decimal representation and the short cut.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Original number:** | 17 | . | 121 | . | 14 | . | 35 |
| **Complement:** | 238 | . | 134 | . | 241 | . | 220 |

### *Bitwise AND Operation*

The bitwise AND operation is a binary operation; it takes two inputs. The AND operation compares the two corresponding bits in two inputs and selects the smaller bit from the two (or select one of them if the bits are equal). Figure 5.3 shows the AND operation.

**Figure 5.3** *Bitwise AND operation*




Although we can directly use the AND operation on the 32-bit binary representation of two numbers, when the numbers are represented in dotted-decimal notation, we can use two short cuts.

1. When at least one of the numbers is 0 or 255, the AND operation selects the smaller byte (or one of them if equal).
2. When none of the two bytes is either 0 or 255, we can write each byte as the sum of eight terms, where each term is a power of 2. We then select the smaller term in each pair (or one of them if equal) and add them to get the result.

### Example 5.8

The following shows how we can apply the AND operation on two 32-bit numbers in binary.

| | | | | |
|---|---|---|---|---|
| **First number:** | 00010001 | 01111001 | 00001110 | 00100011 |
| **Second number:** | 11111111 | 11111111 | 10001100 | 00000000 |
| **Result** | 00010001 | 01111001 | 00001100 | 00000000 |

We can use the same operation using the dotted-decimal representation and the short cut.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **First number:** | 17 | . | 121 | . | 14 | . | 35 |
| **Second number:** | 255 | . | 255 | . | 140 | . | 0 |
| **Result:** | 17 | . | 121 | . | 12 | . | 0 |

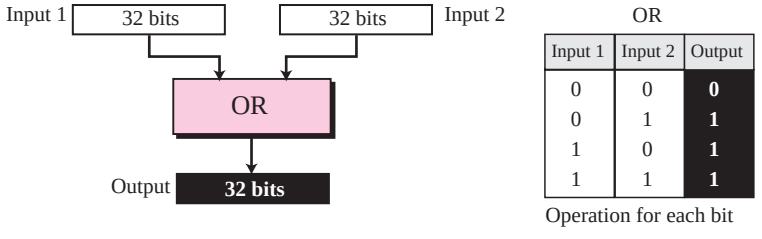
We have applied the first short cut on the first, second, and the fourth byte; we have applied the second short cut on the third byte. We have written 14 and 140 as the sum of terms and selected the smaller term in each pair as shown below.

| **Powers** | $2^7$ | | $2^6$ | | $2^5$ | | $2^4$ | | $2^3$ | | $2^2$ | | $2^1$ | | $2^0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Byte (14)** | 0 | + | 0 | + | 0 | + | 0 | + | 8 | + | 4 | + | 2 | + | 0 |
| **Byte (140)** | 128 | + | 0 | + | 0 | + | 0 | + | 8 | + | 4 | + | 0 | + | 0 |
| **Result (12)** | 0 | + | 0 | + | 0 | + | 0 | + | 8 | + | 4 | + | 0 | + | 0 |

### *Bitwise OR Operation*

The bitwise OR operation is a binary operation; it takes two inputs. The OR operation compares the corresponding bits in the two numbers and selects the larger bit from the two (or one of them if equal). Figure 5.4 shows the OR operation.

**Figure 5.4** *Bitwise OR operation*




OR

| Input 1 | Input 2 | Output |
|---|---|---|
| 0 | 0 | **0** |
| 0 | 1 | **1** |
| 1 | 0 | **1** |
| 1 | 1 | **1** |

Operation for each bit

Although we can directly use the OR operation on the 32-bit binary representation of the two numbers, when the numbers are represented in dotted-decimal notation, we can use two short cuts.

1. When at least one of the two bytes is 0 or 255, the OR operation selects the larger byte (or one of them if equal).
2. When none of the two bytes is 0 or 255, we can write each byte as the sum of eight terms, where each term is a power of 2. We then select the larger term in each pair (or one of them if equal) and add them to get the result of OR operation.

### Example 5.9

The following shows how we can apply the OR operation on two 32-bit numbers in binary.

| | | | | |
|---|---|---|---|---|
| **First number:** | 00010001 | 01111001 | 00001110 | 00100011 |
| **Second number:** | 11111111 | 11111111 | 10001100 | 00000000 |
| **Result** | 11111111 | 11111111 | 10001110 | 00100011 |

We can use the same operation using the dotted-decimal representation and the short cut.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **First number:** | 17 | . | 121 | . | 14 | . | 35 |
| **Second number:** | 255 | . | 255 | . | 140 | . | 0 |
| **Result:** | 255 | . | 255 | . | 142 | . | 35 |

We have used the first short cut for the first and second bytes and the second short cut for the third byte.

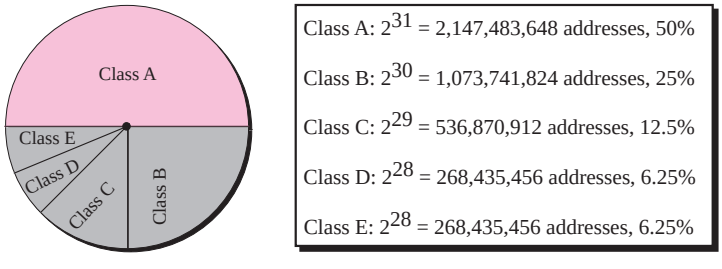
## 5.2 CLASSFUL ADDRESSING

IP addresses, when started a few decades ago, used the concept of *classes*. This architecture is called **classful addressing.** In the mid-1990s, a new architecture, called **classless addressing,** was introduced that supersedes the original architecture. In this section, we introduce classful addressing because it paves the way for understanding classless addressing and justifies the rationale for moving to the new architecture. Classless addressing is discussed in the next section.

### Classes

In classful addressing, the IP address space is divided into five **classes: A, B, C, D,** and **E.** Each class occupies some part of the whole address space. Figure 5.5 shows the class occupation of the address space.

**Figure 5.5** *Occupation of the address space*

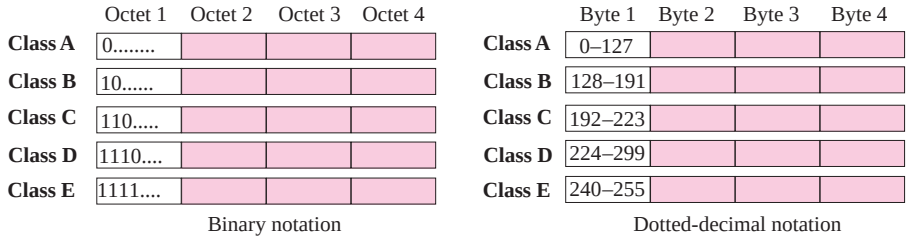



**In classful addressing, the address space is divided into five classes: A, B, C, D, and E.**

### *Recognizing Classes*

We can find the class of an address when the address is given either in binary or dotted-decimal notation. In the binary notation, the first few bits can immediately tell us the class of the address; in the dotted-decimal notation, the value of the first byte can give the class of an address (Figure 5.6).

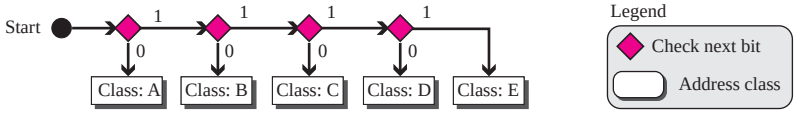
**Figure 5.6** *Finding the class of an address*




Note that some special addresses fall in class A or E. We emphasize that these special addresses are exceptions to the classification; they are discussed later in the chapter.

Computers often store IPv4 addresses in binary notation. In this case, it is very convenient to write an algorithm to use a continuous checking process for finding the address as shown in Figure 5.7.

**Figure 5.7** *Finding the address class using continuous checking*




### Example 5.10

Find the class of each address:

**a.** **00000001 00001011 00001011 11101111**
**b.** **11000001 10000011 00011011 11111111**
**c.** **10100111 11011011 10001011 01101111**
**d.** **11110011 10011011 11111011 00001111**

**Solution**

See the procedure in Figure 5.7.

**a.** The first bit is 0. This is a class A address.
**b.** The first 2 bits are 1; the third bit is 0. This is a class C address.
**c.** The first bit is 1; the second bit is 0. This is a class B address.
**d.** The first 4 bits are 1s. This is a class E address.

### Example 5.11

Find the class of each address:

**a.** 227.12.14.87
**b.** 193.14.56.22
**c.** 14.23.120.8
**d.** 252.5.15.111

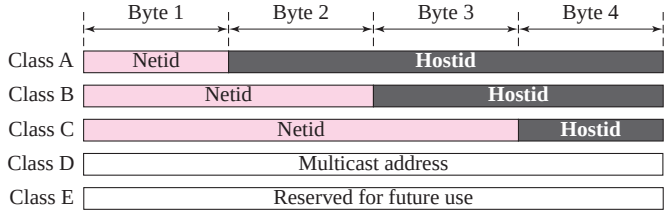
**Solution**

**a.** The first byte is 227 (between 224 and 239); the class is D.
**b.** The first byte is 193 (between 192 and 223); the class is C.
**c.** The first byte is 14 (between 0 and 127); the class is A.
**d.** The first byte is 252 (between 240 and 255); the class is E.

### *Netid and Hostid*

In classful addressing, an IP address in classes A, B, and C is divided into **netid** and **hostid.** These parts are of varying lengths, depending on the class of the address. Figure 5.8 shows the netid and hostid bytes. Note that classes D and E are not divided into netid and hostid, for reasons that we will discuss later.

**Figure 5.8** *Netid and hostid*




In class A, 1 byte defines the netid and 3 bytes define the hostid. In class B, 2 bytes define the netid and 2 bytes define the hostid. In class C, 3 bytes define the netid and 1 byte defines the hostid.

## Classes and Blocks

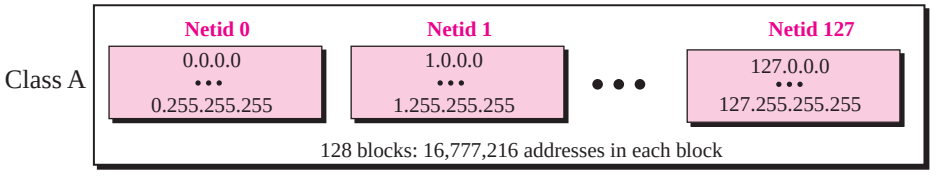
One problem with classful addressing is that each class is divided into a fixed number of blocks with each block having a fixed size. Let us look at each class.

### *Class A*

Since only 1 byte in class A defines the netid and the leftmost bit should be 0, the next 7 bits can be changed to find the number of blocks in this class. Therefore, class A is divided into $2^7 = 128$ blocks that can be assigned to 128 organizations (the number is less because some blocks were reserved as special blocks). However, each block in this class contains 16,777,216 addresses, which means the organization should be a really

large one to use all these addresses. Many addresses are wasted in this class. Figure 5.9 shows the block in class A.

**Figure 5.9** *Blocks in class A*

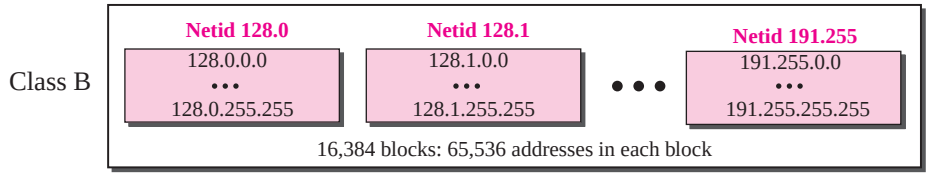



**Millions of class A addresses are wasted.**

### *Class B*

Since 2 bytes in class B define the class and the two leftmost bit should be 10 (fixed), the next 14 bits can be changed to find the number of blocks in this class. Therefore, class B is divided into $2^{14} = 16{,}384$ blocks that can be assigned to 16,384 organizations (the number is less because some blocks were reserved as special blocks). However, each block in this class contains 65,536 addresses. Not so many organizations can use so many addresses. Many addresses are wasted in this class. Figure 5.10 shows the blocks in class B.

**Figure 5.10** *Blocks in class B*

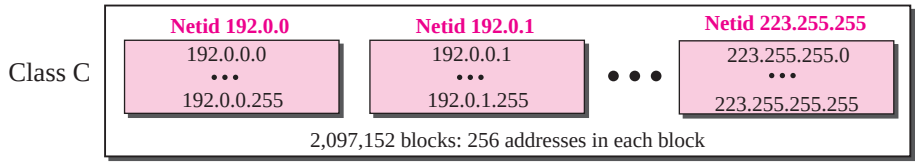



**Many class B addresses are wasted.**

### *Class C*

Since 3 bytes in class C define the class and the three leftmost bits should be 110 (fixed), the next 21 bits can be changed to find the number of blocks in this class. Therefore, class C is divided into $2^{21} = 2{,}097{,}152$ blocks, in which each block contains 256 addresses, that can be assigned to 2,097,152 organizations (the number is less because some blocks were reserved as special blocks). Each block contains 256 addresses. However, not so many organizations were so small as to be satisfied with a class C block. Figure 5.11 shows the blocks in class C.

**Figure 5.11** *Blocks in class C*

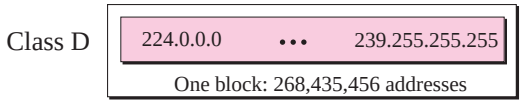



**Not so many organizations are so small to have a class C block.**

### *Class D*

There is just one block of class D addresses. It is designed for multicasting, as we will see in a later section. Each address in this class is used to define one group of hosts on the Internet. When a group is assigned an address in this class, every host that is a member of this group will have a multicast address in addition to its normal (unicast) address. Figure 5.12 shows the block.

**Figure 5.12** *The single block in Class D*

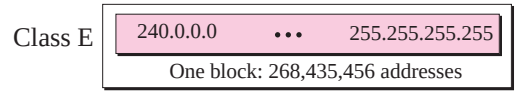



**Class D addresses are made of one block, used for multicasting.**

### *Class E*

There is just one block of class E addresses. It was designed for use as reserved addresses, as shown in Figure 5.13.

**Figure 5.13** *The single block in Class E*




**The only block of class E addresses was reserved for future purposes.**

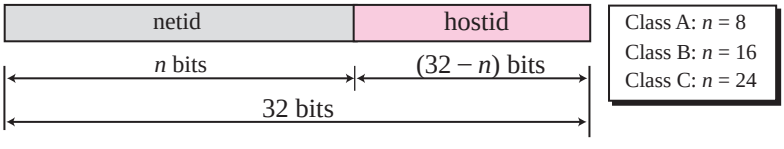
## Two-Level Addressing

The whole purpose of IPv4 addressing is to define a destination for an Internet packet (at the network layer). When classful addressing was designed, it was assumed that the whole Internet is divided into many networks and each network connects many hosts. In other words, the Internet was seen as a network of networks. A network was normally created by an organization that wanted to be connected to the Internet. The Internet authorities allocated a block of addresses to the organization (in class A, B, or C).

**The range of addresses allocated to an organization in classful addressing was a block of addresses in Class A, B, or C.**

Since all addresses in a network belonged to a single block, each address in classful addressing contains two parts: netid and hostid. The netid defines the network; the hostid defines a particular host connected to that network. Figure 5.14 shows an IPv4 address in classful addressing. If *n* bits in the class defines the net, then 32 – *n* bits defines the host. However, the value of *n* depends on the class the block belongs to. The value of *n* can be 8, 16 or 24 corresponding to classes A, B, and C respectively.

**Figure 5.14** *Two-level addressing in classful addressing*




### Example 5.12

Two-level addressing can be found in other communication systems. For example, a telephone system inside the United States can be thought of as two parts: area code and local part. The area code defines the area, the local part defines a particular telephone subscriber in that area.

**(626) 3581301**

The area code, 626, can be compared with the netid, the local part, 3581301, can be compared to the hostid.

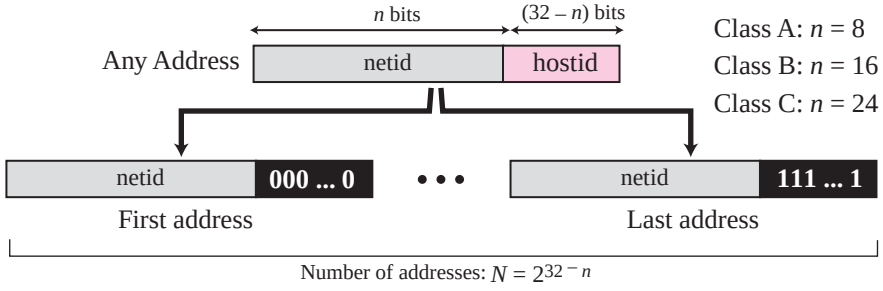
### *Extracting Information in a Block*

A block is a range of addresses. Given any address in the block, we normally like to know three pieces of information about the block: the number of addresses, the first address, and the last address. Before we can extract these pieces of information, we need to know the class of the address, which we showed how to find in the previous section. After the class of the block is found, we know the value of *n*, the length of netid in bits. We can now find these three pieces of information as shown in Figure 5.15.

1. The number of addresses in the block, *N*, can be found using $N = 2^{32-n}$.

2. To find the first address, we keep the $n$ leftmost bits and set the $(32 - n)$ rightmost bits all to 0s.
3. To find the last address, we keep the $n$ leftmost bits and set the $(32 - n)$ rightmost bits all to 1s.

**Figure 5.15** *Information extraction in classful addressing*



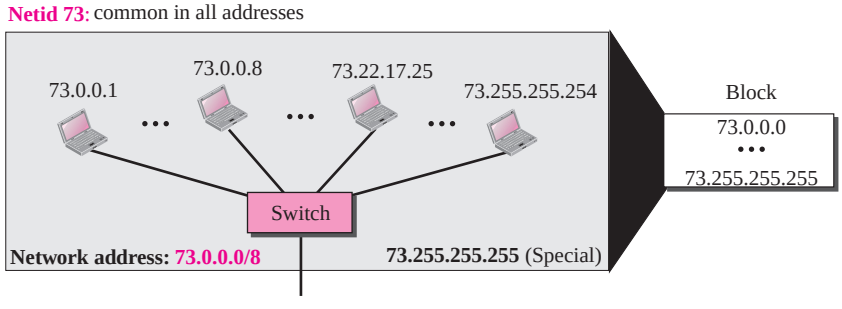

## Example 5.13

An address in a block is given as 73.22.17.25. Find the number of addresses in the block, the first address, and the last address.

### Solution

Since 73 is between 0 and 127, the class of the address is A. The value of $n$ for class A is 8. Figure 5.16 shows a possible configuration of the network that uses this block. Note that we show the value of $n$ in the network address after a slash. Although this was not a common practice in classful addressing, it helps to make it a habit in classless addressing in the next section.

**Figure 5.16** *Solution to Example 5.13*




1. The number of addresses in this block is $N = 2^{32-n} = 2^{24} = 16{,}777{,}216$.
2. To find the first address, we keep the leftmost 8 bits and set the rightmost 24 bits all to 0s. The first address is 73.0.0.0/8 in which 8 is the value of $n$. The first address is called the *network address* and is not assigned to any host. It is used to define the network.

3. To find the last address, we keep the leftmost 8 bits and set the rightmost 24 bits all to 1s. The last address is 73.255.255.255. The last address is normally used for a special purpose, as discussed later in the chapter.

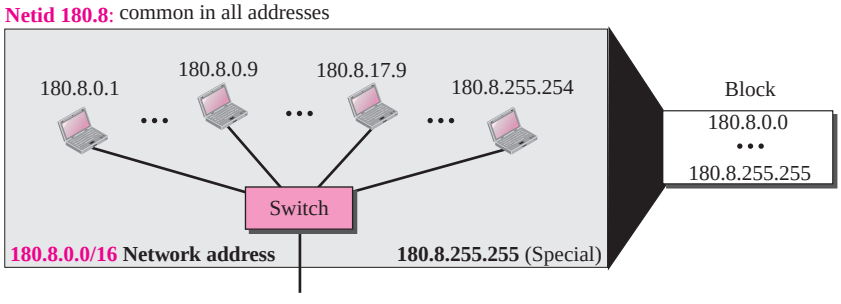
## Example 5.14

An address in a block is given as 180.8.17.9. Find the number of addresses in the block, the first address, and the last address.

### Solution

Since 180 is between 128 and 191, the class of the address is B. The value of *n* for class B is 16. Figure 5.17 shows a possible configuration of the network that uses this block.

**Figure 5.17** *Solution to Example 5.14*




1. The number of addresses in this block is $N = 2^{32-n} = 2^{16} = 65{,}536$.
2. To find the first address, we keep the leftmost 16 bits and set the rightmost 16 bits all to 0s. The first address (network address) is 18.8.0.0/16, in which 16 is the value of *n*.
3. To find the last address, we keep the leftmost 16 bits and set the rightmost 16 bits all to 1s. The last address is 18.8.255.255.

## Example 5.15

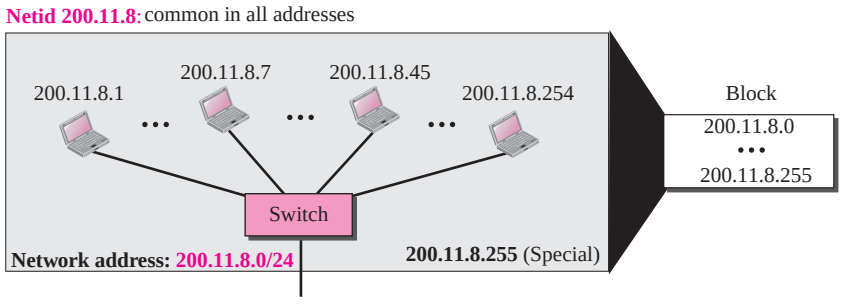
An address in a block is given as 200.11.8.45. Find the number of addresses in the block, the first address, and the last address.

### Solution

Since 200 is between 192 and 223, the class of the address is C. The value of *n* for class C is 24. Figure 5.18 shows a possible configuration of the network that uses this block.

1. The number of addresses in this block is $N = 2^{32-n} = 2^{8} = 256$.
2. To find the first address, we keep the leftmost 24 bits and set the rightmost 8 bits all to 0s. The first address is 200.11.8.0/24. The first address is called the network address.
3. To find the last address, we keep the leftmost 24 bits and set the rightmost 8 bits all to 1s. The last address is 200.11.8.255.

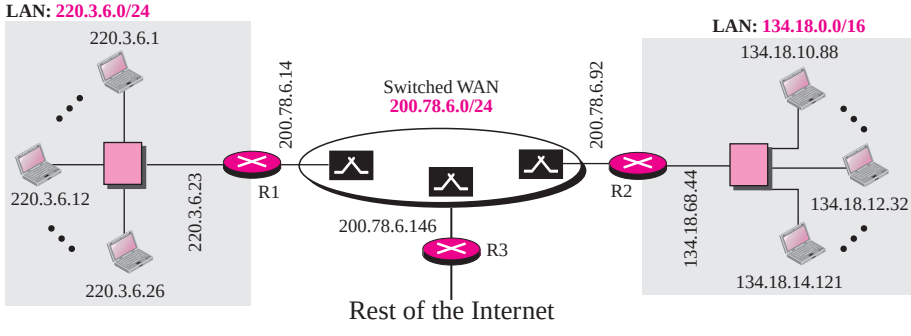
**Figure 5.18** *Solution to Example 5.15*




## An Example

Figure 5.19 shows a hypothetical part of an internet with three networks.

**Figure 5.19** *Sample internet*




We have

1. A LAN with the network address 220.3.6.0 (class C).
2. A LAN with the network address 134.18.0.0 (class B).
3. A switched WAN (class C), such as Frame Relay or ATM, that can be connected to many routers. We have shown three. One router connects the WAN to the left LAN, one connects the WAN to the right LAN, and one connects the WAN to the rest of the internet.

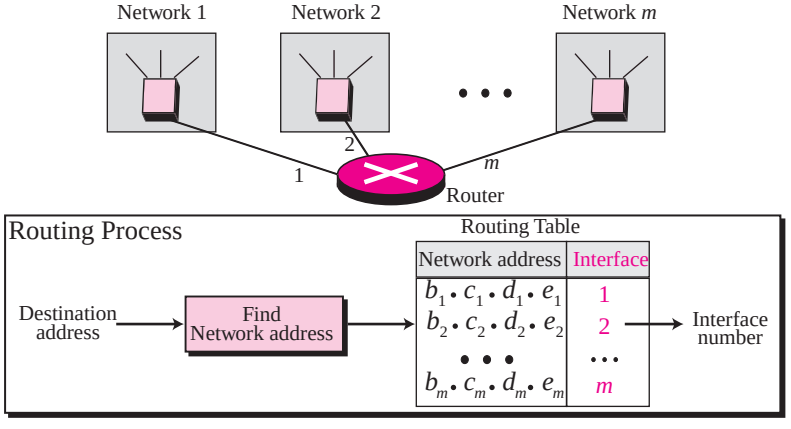
### *Network Address*

The above three examples show that, given any address, we can find all information about the block. The first address, **network address,** is particularly important because it is used in routing a packet to its destination network. For the moment, let us assume that an internet is made of *m* networks and a router with *m* interfaces. When a packet arrives at the router from any source host, the router needs to know to which network the packet

should be sent; the router needs to know from which interface the packet should be sent out. When the packet arrives at the network, it reaches its destination host using another strategy that we discuss in later chapters. Figure 5.20 shows the idea. After the network address has been found, the router consults its routing table to find the corresponding interface from which the packet should be sent out. The network address is actually the identifier of the network; each network is identified by its network address.

**The network address is the identifier of a network.**

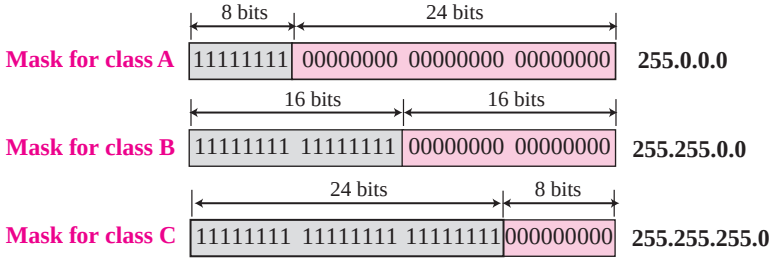
**Figure 5.20** *Network address*




### *Network Mask*

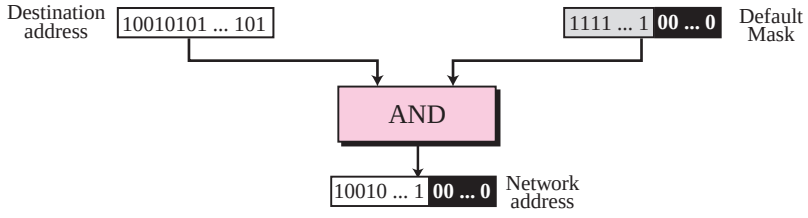
The methods we described previously for extracting the network address are mostly used to show the concept. The routers in the Internet normally use an algorithm to extract the network address from the destination address of a packet. To do this, we need a network mask. A **network mask** or a **default mask** in classful addressing is a 32-bit number with $n$ leftmost bits all set to 1s and $(32 - n)$ rightmost bits all set to 0s. Since $n$ is different for each class in classful addressing, we have three default masks in classful addressing as shown in Figure 5.21.

**Figure 5.21** *Network mask*



To extract the network address from the destination address of a packet, a router uses the AND operation described in the previous section. When the destination address (or any address in the block) is ANDed with the default mask, the result is the network address (Figure 5.22). The router applies the AND operation on the binary (or hexadecimal representation) of the address and the mask, but when we show an example, we use the short cut discussed before and apply the mask on the dotted-decimal notation. The default mask can also be used to find the number of addresses in the block and the last address in the block, but we discuss these applications in classless addressing.

**Figure 5.22** *Finding a network address using the default mask*




### Example 5.16

A router receives a packet with the destination address 201.24.67.32. Show how the router finds the network address of the packet.

**Solution**

We assume that the router first finds the class of the address and then uses the corresponding default mask on the destination address, but we need to know that a router uses another strategy as we will discuss in the next chapter. Since the class of the address is B, we assume that the router applies the default mask for class B, 255.255.0.0 to find the network address.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Destination address** | → | 201 | . | 24 | . | 67 | . | 32 |
| **Default mask** | → | 255 | . | 255 | . | 0 | . | 0 |
| **Network address** | → | 201 | . | 24 | . | 0 | . | 0 |

We have used the first short cut as described in the previous section. The network address is 201.24.0.0 as expected.

## Three-Level Addressing: Subnetting

As we discussed before, the IP addresses were originally designed with two levels of addressing. To reach a host on the Internet, we must first reach the network and then the host. It soon became clear that we need more than two hierarchical levels, for two reasons. First, an organization that was granted a block in class A or B needed to divide its large network into several subnetworks for better security and management. Second, since the blocks in class A and B were almost depleted and the blocks in class C were smaller than the needs of most organizations, an organization that has been granted a block in class A or B could divide the block into smaller subblocks and share them with

other organizations. The idea of splitting a block to smaller blocks is referred to as subnetting. In **subnetting,** a network is divided into several smaller subnetworks (subnets) with each subnetwork having its own subnetwork address.

### Example 5.17

Three-level addressing can be found in the telephone system if we think about the local part of a telephone number as an exchange and a subscriber connection:

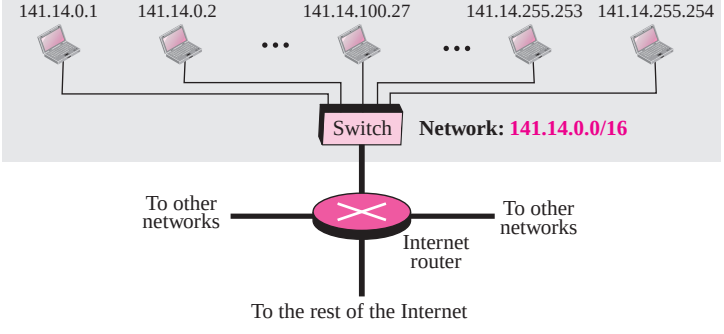
**(626) 358 - 1301**

in which 626 is the area code, 358 is the exchange, and 1301 is the subscriber connection.

### Example 5.18

Figure 5.23 shows a network using class B addresses before subnetting. We have just one network with almost $2^{16}$ hosts. The whole network is connected, through one single connection, to one of the routers in the Internet. Note that we have shown /16 to show the length of the netid (class B).

**Figure 5.23** *Example 5.18*




### Example 5.19

Figure 5.24 shows the same network in Figure 5.23 after subnetting. The whole network is still connected to the Internet through the same router. However, the network has used a private router to divide the network into four subnetworks. The rest of the Internet still sees only one network; internally the network is made of four subnetworks. Each subnetwork can now have almost $2^{14}$ hosts. The network can belong to a university campus with four different schools (buildings). After subnetting, each school has its own subnetworks, but still the whole campus is one network for the rest of the Internet. Note that /16 and /18 show the length of the netid and subnetids.

### *Subnet Mask*

We discussed the network mask (default mask) before. The network mask is used when a network is not subnetted. When we divide a network to several subnetworks, we need to create a subnetwork mask (or subnet mask) for each subnetwork. A subnetwork has subnetid and hostid as shown in Figure 5.25.

**Figure 5.24** *Example 5.19*

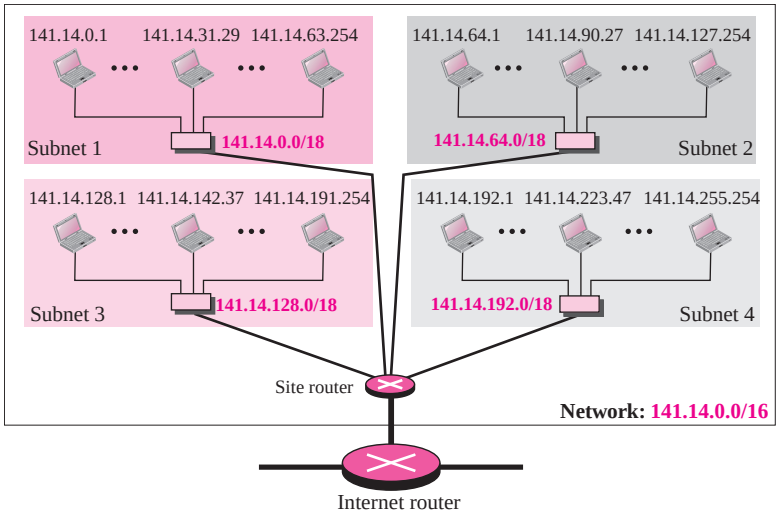
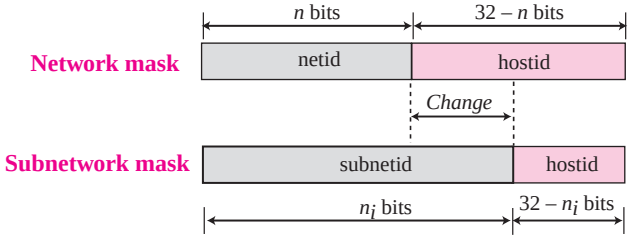



**Figure 5.25** *Network mask and subnetwork mask*




Subnetting increases the length of the netid and decreases the length of hostid. When we divide a network to $s$ number of subnetworks, each of equal numbers of hosts, we can calculate the subnetid for each subnetwork as

$$n_{sub} = n + \log_2 s$$

in which $n$ is the length of netid, $n_{sub}$ is the length of each subnetid, and $s$ is the number of subnets which must be a power of 2.

### Example 5.20

In Example 5.19, we divided a class B network into four subnetworks. The value of $n = 16$ and the value of $n_1 = n_2 = n_3 = n_4 = 16 + \log_2 4 = 18$. This means that the subnet mask has eighteen 1s and fourteen 0s. In other words, the subnet mask is 255.255.192.0 which is different from the network mask for class B (255.255.0.0).

### *Subnet Address*

When a network is subnetted, the first address in the subnet is the identifier of the subnet and is used by the router to route the packets destined for that subnetwork. Given any address in the subnet, the router can find the subnet mask using the same procedure we discussed to find the network mask: ANDing the given address with the subnet mask. The short cuts we discussed in the previous section can be used to find the subnet address.

### Example 5.21

In Example 5.19, we show that a network is divided into four subnets. Since one of the addresses in subnet 2 is 141.14.120.77, we can find the subnet address as:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Address** | → | 141 | . | 14 | . | 120 | . | 77 |
| **Mask** | → | 255 | . | 255 | . | 192 | . | 0 |
| **Subnet Address** | → | 141 | . | 14 | . | 64 | . | 0 |

The values of the first, second, and fourth bytes are calculated using the first short cut for AND operation. The value of the third byte is calculated using the second short cut for the AND operation.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Address (120)** | 0 | + | 64 | + | 32 | + | 16 | + | 8 | + | 0 | + | 0 | + | 0 |
| **Mask (192)** | 128 | + | 64 | + | 0 | + | 0 | + | 0 | + | 0 | + | 0 | + | 0 |
| **Result (64)** | 0 | + | 64 | + | 0 | + | 0 | + | 0 | + | 0 | + | 0 | + | 0 |

### *Designing Subnets*

We show how to design a subnet when we discuss classless addressing. Since classful addressing is a special case of classless addressing, what is discussed later can also be applied to classful addressing.

## Supernetting

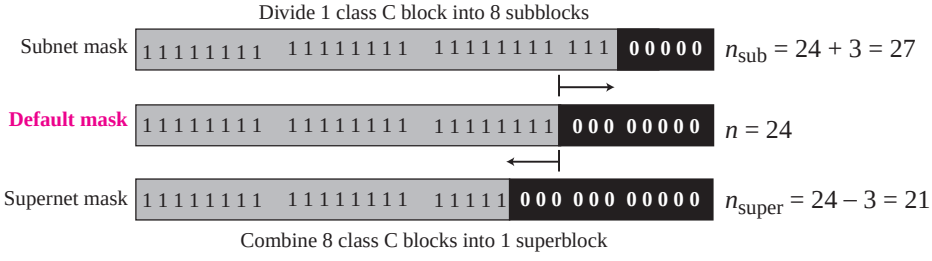
Subnetting could not completely solve address depletion problems in classful addressing because most organizations did not want to share their granted blocks with others. Since class C blocks were still available but the size of the block did not meet the requirement of new organizations that wanted to join the Internet, one solution was **supernetting.** In supernetting, an organization can combine several class C blocks to create a larger range of addresses. In other words, several networks are combined to create a supernetwork. By doing this, an organization can apply for several class C blocks instead of just one. For example, an organization that needs 1000 addresses can be granted four class C blocks.

### *Supernet Mask*

A **supernet mask** is the reverse of a subnet mask. A subnet mask for class C has more 1s than the default mask for this class. A supernet mask for class C has less 1s than the default mask for this class.

Figure 5.26 shows the difference between a subnet mask and a supernet mask. A subnet mask that divides a block into eight subblocks has three more 1s ($2^3 = 8$) than the default mask; a supernet mask that combines eight blocks into one superblock has three less 1s than the default mask.

**Figure 5.26** *Comparison of subnet, default, and supernet masks*




In supernetting, the number of class C addresses that can be combined to make a supernet needs to be a power of 2. The length of the supernetid can be found using the formula

$$n_{super} = n - \log_2 c$$

in which $n_{super}$ defines the length of the supernetid in bits and $c$ defines the number of class C blocks that are combined.

Unfortunately, supernetting provided two new problems: First, the number of blocks to combine needs to be a power of 2, which means an organization that needed seven blocks should be granted at least eight blocks (address wasting). Second, supernetting and subnetting really complicated the routing of packets in the Internet.

## 5.3 CLASSLESS ADDRESSING

Subnetting and supernetting in classful addressing did not really solve the address depletion problem and made the distribution of addresses and the routing process more difficult. With the growth of the Internet, it was clear that a larger address space was needed as a long-term solution. The larger address space, however, requires that the length of IP addresses to be increased, which means the format of the IP packets needs to be changed. Although the long-range solution has already been devised and is called IPv6 (see Chapters 26 to 28), a short-term solution was also devised to use the same address space but to change the distribution of addresses to provide a fair share to each organization. The short-term solution still uses IPv4 addresses, but it is called *classless* addressing. In other words, the class privilege was removed from the distribution to compensate for the address depletion.

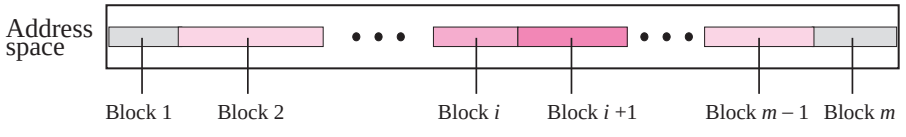
There was another motivation for classless addressing. During the 1990s, Internet service providers (ISPs) came into prominence. An ISP is an organization that provides Internet access for individuals, small businesses, and midsize organizations that do not want to create an Internet site and become involved in providing Internet services (such as e-mail services) for their employees. An ISP can provide these services. An ISP is granted a large range of addresses and then subdivides the addresses (in groups of 1, 2, 4, 8, 16, and so on), giving a range of addresses to a household or a small business. The customers are connected via a dial-up modem, DSL, or cable modem to the ISP. However, each customer needs some IPv4 addresses.

In 1996, the Internet authorities announced a new architecture called classless addressing. In classless addressing, variable-length blocks are used that belong to no classes. We can have a block of 1 address, 2 addresses, 4 addresses, 128 addresses, and so on.

## Variable-Length Blocks

In classful addressing the whole address space was divided into five classes. Although each organization was granted one block in class A, B, or C, the size of the blocks was predefined; the organization needed to choose one of the three block sizes. The only block in class D and the only block in class E were reserved for a special purpose. In classless addressing, the whole address space is divided into variable length blocks. Theoretically, we can have a block of $2^0, 2^1, 2^2, \ldots, 2^{32}$ addresses. The only restriction, as we discuss later, is that the number of addresses in a block needs to be a power of 2. An organization can be granted one block of addresses. Figure 5.27 shows the division of the whole address space into nonoverlapping blocks.

**Figure 5.27** *Variable-length blocks in classless addressing*

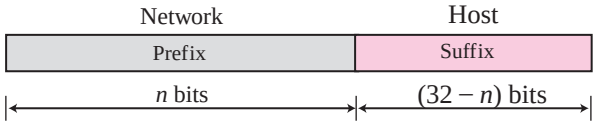



## Two-Level Addressing

In classful addressing, two-level addressing was provided by dividing an address into *netid* and *hostid*. The netid defined the network; the hostid defined the host in the network. The same idea can be applied in classless addressing. When an organization is granted a block of addresses, the block is actually divided into two parts, the **prefix** and the **suffix.** The prefix plays the same role as the netid; the suffix plays the same role as the hostid. All addresses in the block have the same prefix; each address has a different suffix. Figure 5.28 shows the prefix and suffix in a classless block.

**In classless addressing, the prefix defines the network and the suffix defines the host.**

**Figure 5.28** *Prefix and suffix*




In classful addressing, the length of the netid, *n*, depends on the class of the address; it can be only 8, 16, or 24. In classless addressing, the length of the prefix, *n*, depends on the size of the block; it can be 0, 1, 2, 3, . . . , 32. In classless addressing, the value of *n* is referred to as **prefix length;** the value of 32 – *n* is referred to as **suffix length.**

**The prefix length in classless addressing can be 1 to 32.**

### Example 5.22

What is the prefix length and suffix length if the whole Internet is considered as one single block with 4,294,967,296 addresses?

**Solution**

In this case, the prefix length is 0 and the suffix length is 32. All 32 bits vary to define $2^{32}$ = 4,294,967,296 hosts in this single block.

### Example 5.23

What is the prefix length and suffix length if the Internet is divided into 4,294,967,296 blocks and each block has one single address?

**Solution**

In this case, the prefix length for each block is 32 and the suffix length is 0. All 32 bits are needed to define $2^{32}$ = 4,294,967,296 blocks. The only address in each block is defined by the block itself.

### Example 5.24

The number of addresses in a block is inversely related to the value of the prefix length, *n*. A small *n* means a larger block; a large *n* means a small block.

### *Slash Notation*

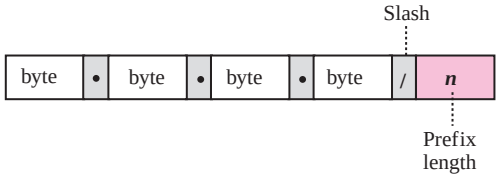
The netid length in classful addressing or the prefix length in classless addressing play a very important role when we need to extract the information about the block from a given address in the block. However, there is a difference here in classful and classless addressing.

- In classful addressing, the netid length is inherent in the address. Given an address, we know the class of the address that allows us to find the netid length (8, 16, or 24).

- In classless addressing, the prefix length cannot be found if we are given only an address in the block. The given address can belong to a block with any prefix length.

In classless addressing, we need to include the prefix length to each address if we need to find the block of the address. In this case, the prefix length, *n*, is added to the address separated by a slash. The notation is informally referred to as **slash notation.** An address in classless addressing can then be represented as shown in Figure 5.29.

**Figure 5.29** *Slash notation*




The slash notation is formally referred to as **classless interdomain routing** or **CIDR** (pronounced cider) notation.

**In classless addressing, we need to know one of the addresses in the block and the prefix length to define the block.**

### Example 5.25

In classless addressing, an address cannot per se define the block the address belongs to. For example, the address 230.8.24.56 can belong to many blocks some of them are shown below with the value of the prefix associated with that block:

| | | | | | |
|---|---|---|---|---|---|
| Prefix length:16 | → | Block: | 230.8.0.0 | to | 230.8.255.255 |
| Prefix length:20 | → | Block: | 230.8.16.0 | to | 230.8.31.255 |
| Prefix length:26 | → | Block: | 230.8.24.0 | to | 230.8.24.63 |
| Prefix length:27 | → | Block: | 230.8.24.32 | to | 230.8.24.63 |
| Prefix length:29 | → | Block: | 230.8.24.56 | to | 230.8.24.63 |
| Prefix length:31 | → | Block: | 230.8.24.56 | to | 230.8.24.57 |

### *Network Mask*

The idea of network mask in classless addressing is the same as the one in classful addressing. A network mask is a 32-bit number with the *n* leftmost bits all set to 0s and the rest of the bits all set to 1s.

### Example 5.26

The following addresses are defined using slash notations.

**a.** In the address 12.23.24.78/8, the network mask is 255.0.0.0. The mask has eight 1s and twenty-four 0s. The prefix length is 8; the suffix length is 24.

**b.** In the address 130.11.232.156/16, the network mask is 255.255.0.0. The mask has sixteen 1s and sixteen 0s.The prefix length is 16; the suffix length is 16.

**c.** In the address 167.199.170.82/27, the network mask is 255.255.255.224. The mask has twenty-seven 1s and five 0s. The prefix length is 27; the suffix length is 5.

### *Extracting Block Information*

An address in slash notation (CIDR) contains all information we need about the block: the first address (network address), the number of addresses, and the last address. These three pieces of information can be found as follows:

- The number of addresses in the block can be found as:

$$N = 2^{32-n}$$

in which $n$ is the prefix length and $N$ is the number of addresses in the block.

- The first address (network address) in the block can be found by ANDing the address with the network mask:

**First address = (any address) AND (network mask)**

Alternatively, we can keep the $n$ leftmost bits of any address in the block and set the 32– $n$ bits to 0s to find the first address.

- The last address in the block can be found by either adding the first address with the number of addresses or, directly, by ORing the address with the complement (NOTing) of the network mask:

**Last address = (any address) OR [NOT (network mask)]**

Alternatively, we can keep the $n$ leftmost bits of any address in the block and set the 32 – $n$ bits to 1s to find the last address.

### Example 5.27

One of the addresses in a block is 167.199.170.82/27. Find the number of addresses in the network, the first address, and the last address.

**Solution**

The value of $n$ is 27. The network mask has twenty-seven 1s and five 0s. It is 255.255.255.240.

**a.** The number of addresses in the network is $2^{32-n} = 2^{32-n} = 2^5 = 32$.

**b.** We use the AND operation to find the first address (network address).The first address is 167.199.170.64/27.

| | |
|---|---|
| Address in binary: | **10100111 11000111 10101010 01010010** |
| Network mask: | **11111111 11111111 11111111 11100000** |
| First address: | **10100111 11000111 10101010 01000000** |

**c.** To find the last address, we first find the complement of the network mask and then OR it with the given address: The last address is 167.199.170.95/27.

| | | | | |
|---|---|---|---|---|
| Address in binary: | 10100111 | 11000111 | 10101010 | 01010010 |
| Complement of network mask: | 00000000 | 00000000 | 00000000 | 00011111 |
| Last address: | 10100111 | 11000111 | 10101010 | 01011111 |

### Example 5.28

One of the addresses in a block is 17.63.110.114/24. Find the number of addresses, the first address, and the last address in the block.

**Solution**

The network mask is 255.255.255.0.

**a.** The number of addresses in the network is $2^{32-24} = 256$.

**b.** To find the first address, we use the short cut methods discussed early in the chapter.

| | | | | |
|---|---|---|---|---|
| Address: | 17 | 63 | 110 | 114 |
| Network mask: | 255 | 255 | 255 | 0 |
| First address (AND): | 17 | 63 | 110 | 0 |

The first address is 17.63.110.0/24.

**c.** To find the last address, we use the complement of the network mask and the first short cut method we discussed before. The last address is 17.63.110.255/24.

| | | | | |
|---|---|---|---|---|
| Address: | 17 | 63 | 110 | 114 |
| Complement of the mask (NOT): | 0 | 0 | 0 | 255 |
| Last address (OR): | 17 | 63 | 110 | 255 |

### Example 5.29

One of the addresses in a block is 110.23.120.14/20. Find the number of addresses, the first address, and the last address in the block.

**Solution**

The network mask is 255.255.240.0.

**a.** The number of addresses in the network is $2^{32-20} = 4096$.

**b.** To find the first address, we apply the first short cut to bytes 1, 2, and 4 and the second short cut to byte 3. The first address is 110.23.112.0/20.

| | | | | |
|---|---|---|---|---|
| Address: | 110 | 23 | 120 | 14 |
| Network mask: | 255 | 255 | 240 | 0 |
| First address (AND): | 110 | 23 | 112 | 0 |

**c.** To find the last address, we apply the first short cut to bytes 1, 2, and 4 and the second short cut to byte 3. The OR operation is applied to the complement of the mask. The last address is 110.23.127.255/20.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Address: | 110 | . | 23 | . | 120 | . | 14 |
| Network mask: | 0 | . | 0 | . | 15 | . | 255 |
| Last address (OR): | 110 | . | 23 | . | 127 | . | 255 |

## Block Allocation

The next issue in classless addressing is block allocation. How are the blocks allocated? The ultimate responsibility of block allocation is given to a global authority called the Internet Corporation for Assigned Names and Addresses (ICANN). However, ICANN does not normally allocate addresses to individual Internet users. It assigns a large block of addresses to an ISP (or a larger organization that is considered an ISP in this case). For the proper operation of the CIDR, three restrictions need to be applied to the allocated block.

1. The number of requested addresses, $N$, needs to be a power of 2. This is needed to provide an integer value for the prefix length, $n$ (see the second restriction). The number of addresses can be 1, 2, 4, 8, 16, and so on.
2. The value of prefix length can be found from the number of addresses in the block. Since $N = 2^{32 - n}$, then $n = \log_2 (2^{32}/N) = 32 - \log_2 N$. That is the reason why $N$ needs to be a power of 2.
3. The requested block needs to be allocated where there are a contiguous number of unallocated addresses in the address space. However, there is a restriction on choosing the beginning addresses of the block. The beginning address needs to be divisible by the number of addresses in the block. To see this restriction, we can show that the beginning address can be calculated as $X \times 2^{n - 32}$ in which $X$ is the decimal value of the prefix. In other words, the beginning address is $X \times N$.

### Example 5.30

An ISP has requested a block of 1000 addresses. The following block is granted.

a. Since 1000 is not a power of 2, 1024 addresses are granted ($1024 = 2^{10}$).
b. The prefix length for the block is calculated as $n = 32 - \log_2 1024 = 22$.
c. The beginning address is chosen as 18.14.12.0 (which is divisible by 1024).

The granted block is 18.14.12.0/22. The first address is 18.14.12.0/22 and the last address is 18.14.15.255/22.

### *Relation to Classful Addressing*

All issues discussed for classless addressing can be applied to classful addressing. As a matter of fact, classful addressing is a special case of the classless addressing in which the blocks in class A, B, and C have the prefix length $n_A = 8$, $n_B = 16$, and $n_C = 24$. A block in classful addressing can be easily changed to a block in class addressing if we use the prefix length defined in Table 5.1.

**Table 5.1** *Prefix length for classful addressing*

| *Class* | *Prefix length* | *Class* | *Prefix length* |
|---|---|---|---|
| A | /8 | D | /4 |
| B | /16 | E | /4 |
| C | /24 | | |

### Example 5.31

Assume an organization has given a class A block as 73.0.0.0 in the past. If the block is not revoked by the authority, the classless architecture assumes that the organization has a block 73.0.0.0/8 in classless addressing.

## Subnetting

Three levels of hierarchy can be created using subnetting. An organization (or an ISP) that is granted a range of addresses may divide the range into several subranges and assign each subrange to a **subnetwork** (or **subnet**). The concept is the same as we discussed for classful addressing. Note that nothing stops the organization from creating more levels. A subnetwork can be divided into several sub-subnetworks. A sub-subnetwork can be divided into several sub-sub-subnetworks. And so on.

### *Designing Subnets*

The subnetworks in a network should be carefully designed to enable the routing of packets. We assume the total number of addresses granted to the organization is $N$, the prefix length is $n$, the assigned number of addresses to each subnetwork is $N_{sub}$, the prefix length for each subnetwork is $n_{sub}$, and the total number of subnetworks is $s$. Then, the following steps need to be carefully followed to guarantee the proper operation of the subnetworks.

1. The number of addresses in each subnetwork should be a power of 2.
2. The prefix length for each subnetwork should be found using the following formula:

$$n_{sub} = n + \log_2 (N/N_{sub})$$

3. The starting address in each subnetwork should be divisible by the number of addresses in that subnetwork. This can be achieved if we first assign addresses to larger networks.

**The restrictions applied in allocating addresses for a subnetwork are parallel to the ones used to allocate addresses for a network.**

### *Finding Information about Each Subnetwork*

After designing the subnetworks, the information about each subnetwork, such as first and last address, can be found using the process we described to find the information about each network in the Internet.

### Example 5.32

An organization is granted the block 130.34.12.64/26. The organization needs four subnetworks, each with an equal number of hosts. Design the subnetworks and find the information about each network.

**Solution**

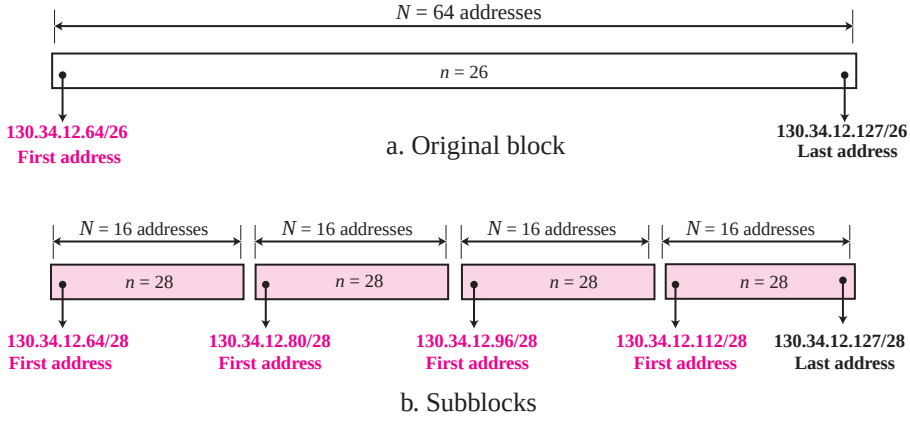
The number of addresses for the whole network can be found as $N = 2^{32 - 26} = 64$. Using the process described in the previous section, the first address in the network is 130.34.12.64/26 and the last address is 130.34.12.127/26. We now design the subnetworks:

1. We grant 16 addresses for each subnetwork to meet the first requirement (64/16 is a power of 2).
2. The **subnetwork** mask for each subnetwork is:

$$n_1 = n_2 = n_3 = n_4 = n + \log_2 (N/N_i) = 26 + \log_2 4 = 28$$

3. We grant 16 addresses to each subnet starting from the first available address. Figure 5.30 shows the subblock for each subnet. Note that the starting address in each subnetwork is divisible by the number of addresses in that subnetwork.

**Figure 5.30** *Solution to Example 5.32*




## Example 5.33

An organization is granted a block of addresses with the beginning address 14.24.74.0/24. The organization needs to have 3 subblocks of addresses to use in its three subnets as shown below:

- One subblock of 120 addresses.
- One subblock of 60 addresses.
- One subblock of 10 addresses.

**Solution**

There are $2^{32 - 24} = 256$ addresses in this block. The first address is 14.24.74.0/24; the last address is 14.24.74.255/24.

**a.** The number of addresses in the first subblock is not a power of 2. We allocate 128 addresses. The first can be used as network address and the last as the special address. There are still 126 addresses available. The subnet mask for this subnet can be found as $n_1 = 24 + \log_2 (256/128) = 25$. The first address in this block is 14.24.74.0/25; the last address is 14.24.74.127/25.

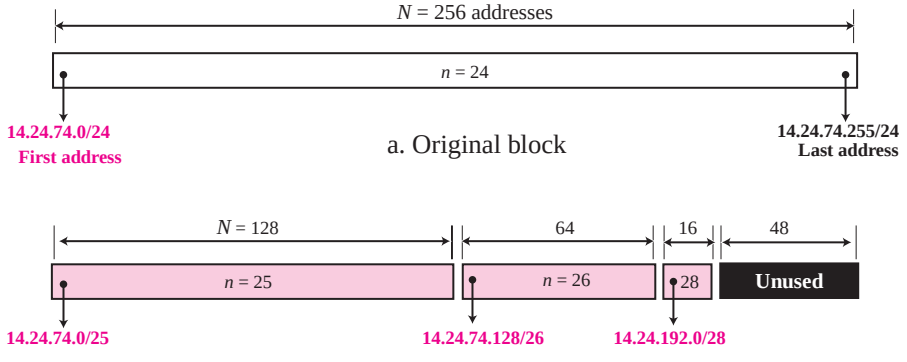
**b.** The number of addresses in the second subblock is not a power of 2 either. We allocate 64 addresses. The first can be used as network address and the last as the special address. There are still 62 addresses available. The subnet mask for this subnet can be found as $n_1 = 24 + \log_2 (256/64) = 26$. The first address in this block is 14.24.74.128/26; the last address is 14.24.74.191/26.

**c.** The number of addresses in the third subblock is not a power of 2 either. We allocate 16 addresses. The first can be used as network address and the last as the special address. There are still 14 addresses available. The subnet mask for this subnet can be found as $n_1 = 24 + \log_2 (256/16) = 28$. The first address in this block is 14.24.74.192/28; the last address is 14.24.74.207/28.

**d.** If we add all addresses in the previous subblocks, the result is 208 addresses, which means 48 addresses are left in reserve. The first address in this range is 14.24.74.209. The last address is 14.24.74.255. We don't know about the prefix length yet.

**e.** Figure 5.31 shows the configuration of blocks. We have shown the first address in each block.

**Figure 5.31** *Solution to Example 5.33*




a. Original block

b. Subblocks

## Example 5.34

Assume a company has three offices: Central, East, and West. The Central office is connected to the East and West offices via private, point-to-point WAN lines. The company is granted a block of 64 addresses with the beginning address 70.12.100.128/26. The management has decided to allocate 32 addresses for the Central office and divides the rest of addresses between the two other offices.

**1.** The number of addresses are assigned as follows:

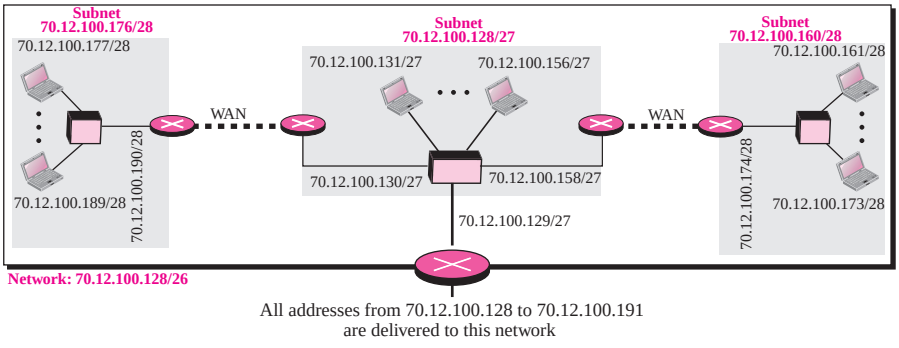
| Central office $N_c = 32$ | East office $N_e = 16$ | West office $N_w = 16$ |
|---|---|---|

**2.** We can find the prefix length for each subnetwork:

| $n_c = n + \log_2(64/32) = 27$ | $n_e = n + \log_2(64/16) = 28$ | $n_w = n + \log_2(64/16) = 28$ |
|---|---|---|

3. Figure 5.32 shows the configuration designed by the management. The Central office uses addresses 70.12.100.128/27 to 70.12.100.159/27. The company has used three of these addresses for the routers and has reserved the last address in the subblock. The East office uses the addresses 70.12.100.160/28 to 70.12.100.175/28. One of these addresses is used for the router and the company has reserved the last address in the subblock. The West office uses the addresses 70.12.100.160/28 to 70.12.100.175/28. One of these addresses is used for the router and the company has reserved the last address in the subblock. The company uses no address for the point-to-point connections in WANs.

**Figure 5.32** *Example 14*




All addresses from 70.12.100.128 to 70.12.100.191 are delivered to this network

### *Address Aggregation*

One of the advantages of CIDR architecture is **address aggregation.** ICANN assigns a large **block of addresses** to an ISP. Each ISP in turn divides its assigned block into smaller subblocks and grants the subblocks to its customers; many blocks of addresses are aggregated in one block and granted to one ISP.

### Example 5.35

An ISP is granted a block of addresses starting with 190.100.0.0/16 (65,536 addresses). The ISP needs to distribute these addresses to three groups of customers as follows:

- The first group has 64 customers; each needs approximately 256 addresses.
- The second group has 128 customers; each needs approximately 128 addresses.
- The third group has 128 customers; each needs approximately 64 addresses.

We design the subblocks and find out how many addresses are still available after these allocations.

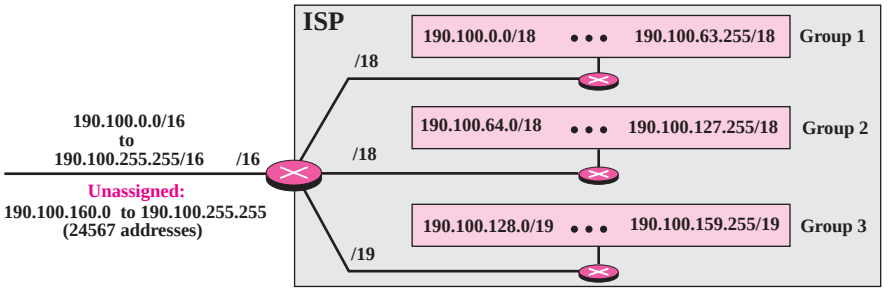
#### Solution

Let us solve the problem in two steps. In the first step, we allocate a subblock of addresses to each group. The total number of addresses allocated to each group and the prefix length for each subblock can found as

| | |
|---|---|
| **Group 1: $64 \times 256 = 16{,}384$** | $n_1 = 16 + \log_2 (65536/16384) = 18$ |
| **Group 2: $128 \times 128 = 16{,}384$** | $n_2 = 16 + \log_2 (65536/16384) = 18$ |
| **Group 3: $128 \times 64 = 8192$** | $n_3 = 16 + \log_2 (65536/8192) = 19$ |

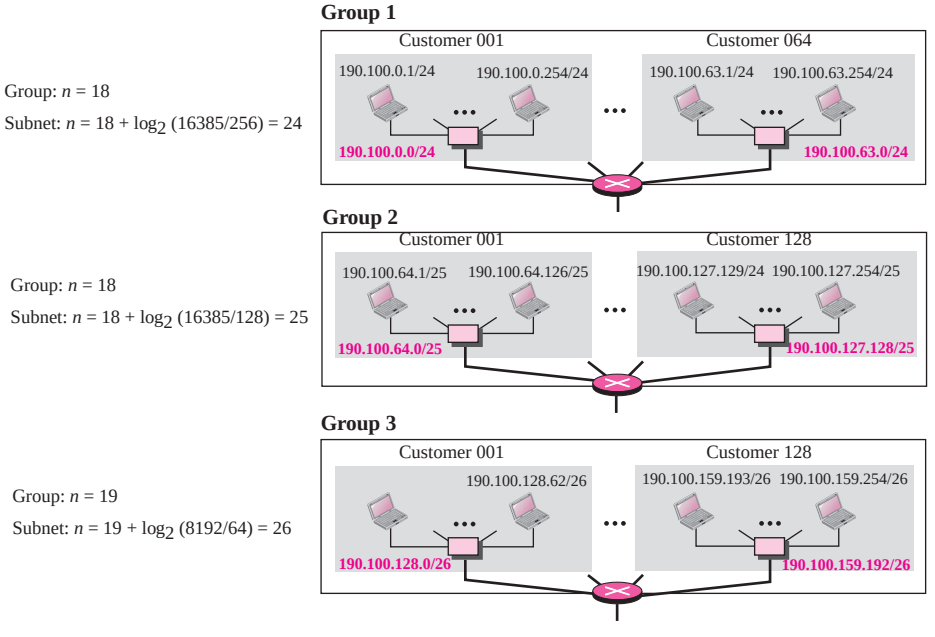
Figure 5.33 shows the design for the first hierarchical level.

**Figure 5.33** *Solution to Example 5.35: first step*




Now we can think about each group. The prefix length changes for the networks in each group depending on the number of addresses used in each network. Figure 5.34 shows the second level of the hierarchy. Note that we have used the first address for each customer as the subnet address and have reserved the last address as a special address.

**Figure 5.34** *Solution to Example 5.35: second step*



## 5.4 SPECIAL ADDRESSES

In classful addressing some addresses were reserved for special purposes. The classless addressing scheme inherits some of these special addresses from classful addressing.

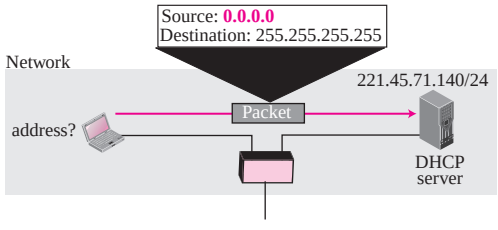
### Special Blocks

Some blocks of addresses are reserved for special purposes.

#### *All-Zeros Address*

The block 0.0.0.0/32, which contains only one single address, is reserved for communication when a host needs to send an IPv4 packet but it does not know its own address. This is normally used by a host at bootstrap time when it does not know its IPv4 address. The host sends an IPv4 packet to a bootstrap server (called DHCP server as discussed in Chapter 18) using this address as the source address and a **limited broadcast address** as the destination address to find its own address (see Figure 5.35).

**Figure 5.35** *Examples of using the all-zeros address*




#### *All-Ones Address: Limited Broadcast Address*

The block 255.255.255.255/32, which contains one single address, is reserved for limited broadcast address in the current network. A host that wants to send a message to every other host can use this address as a destination address in an IPv4 packet. However, a router will block a packet having this type of address to confine the broadcasting to the local network. In Figure 5.36, a host sends a datagram using a destination IPv4 address consisting of all 1s. All devices on this network receive and process this datagram.

#### *Loopback Addresses*

The block 127.0.0.0/8 is used for the **loopback address,** which is an address used to test the software on a machine. When this address is used, a packet never leaves the machine; it simply returns to the protocol software. It can be used to test the IPv4 software. For example, an application such as "ping" can send a packet with a loopback address as the destination address to see if the IPv4 software is capable of receiving and processing a packet. As another example, the loopback address can be used by a *client process* (a running application program) to send a message to a server process on the same machine. Note that this can be used only as a destination address in an IPv4 packet (see Figure 5.37).

**Figure 5.36** *Example of limited broadcast address*

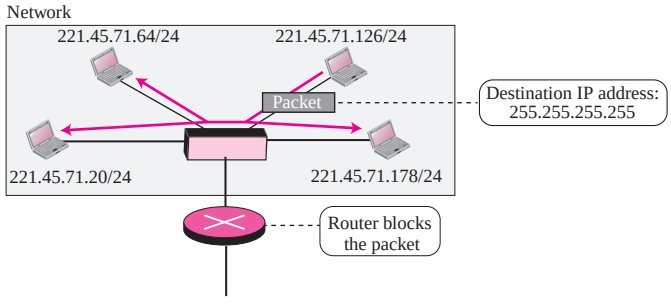
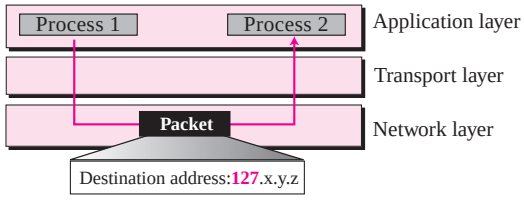



**Figure 5.37** *Example of loopback address*




### *Private Addresses*

A number of blocks are assigned for private use. They are not recognized globally. These blocks are depicted in Table 5.2. These addresses are used either in isolation or in connection with network address translation techniques (see NAT section later in this chapter).

**Table 5.2** *Addresses for private networks*

| *Block* | *Number of addresses* | *Block* | *Number of addresses* |
|---|---|---|---|
| 10.0.0.0/8 | 16,777,216 | 192.168.0.0/16 | 65,536 |
| 172.16.0.0/12 | 1,047,584 | 169.254.0.0/16 | 65,536 |

### *Multicast Addresses*

The block 224.0.0.0/4 is reserved for multicast communication. We discuss multicasting in Chapter 12 in detail.

## Special Addresses in Each block

It is not mandatory, but it is recommended, that some addresses in a block be used for special addresses. These addresses are not assigned to any host. However, if a block (or subblock) is so small, we cannot afford to use part of the addresses as special addresses.

### *Network Address*

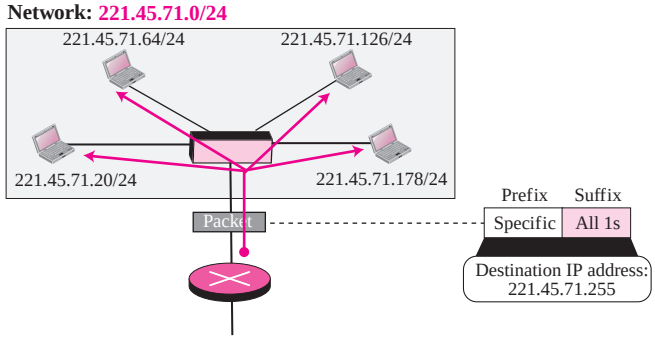
We have already discussed network addresses. The first address (with the suffix set all to 0s) in a block defines the network address. It actually defines the network itself

(cabling) and not any host in the network. Of course, the first address in a subnetwork is called the subnetwork address and plays the same role.

### *Direct Broadcast Address*

The last address in a block or subblock (with the suffix set all to 1s) can be used as a **direct broadcast address.** This address is usually used by a router to send a packet to all hosts in a specific network. All hosts will accept a packet having this type of destination address. Note that this address can be used only as a destination address in an IPv4 packet. In Figure 5.38, the router sends a datagram using a destination IPv4 address with a suffix of all 1s. All devices on this network receive and process the datagram.

**Figure 5.38** *Example of a direct broadcast address*



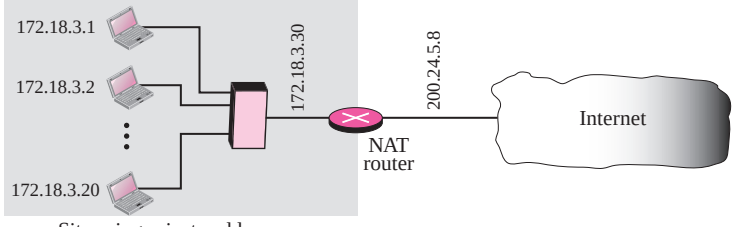

## 5.5 NAT

The distribution of addresses through ISPs has created a new problem. Assume that an ISP has granted a small range of addresses to a small business or a household. If the business grows or the household needs a larger range, the ISP may not be able to grant the demand because the addresses before and after the range may have already been allocated to other networks. In most situations, however, only a portion of computers in a small network need access to the Internet simultaneously. This means that the number of allocated addresses does not have to match the number of computers in the network. For example, assume a small business with 20 computers in which the maximum number of computers that access the Internet simultaneously is only 5. Most of the computers are either doing some task that does not need Internet access or communicating with each other. This small business can use the TCP/IP protocol for both internal and universal communication. The business can use 20 (or 25) addresses from the private block addresses discussed before for internal communication; five addresses for universal communication can be assigned by the ISP.

A technology that can provide the mapping between the private and universal addresses, and at the same time, support virtual private networks that we discuss in Chapter 30, is **network address translation (NAT).** The technology allows a site to use a set of private addresses for internal communication and a set of global Internet addresses

(at least one) for communication with the rest of the world. The site must have only one single connection to the global Internet through a NAT-capable router that runs NAT software. Figure 5.39 shows a simple implementation of NAT.

**Figure 5.39** *NAT*

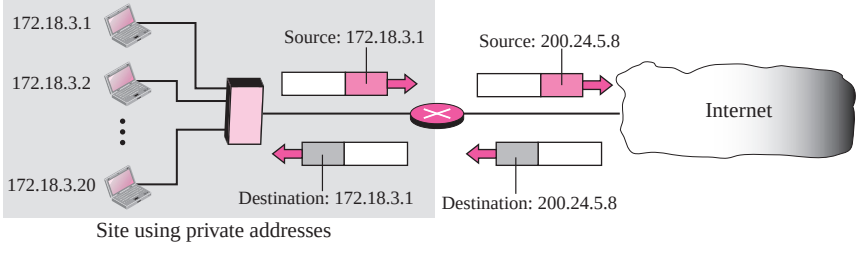



As the figure shows, the private network uses private addresses. The router that connects the network to the global address uses one private address and one global address. The private network is transparent to the rest of the Internet; the rest of the Internet sees only the NAT router with the address 200.24.5.8.

## Address Translation

All of the outgoing packets go through the NAT router, which replaces the *source address* in the packet with the global NAT address. All incoming packets also pass through the NAT router, which replaces the *destination address* in the packet (the NAT router global address) with the appropriate private address. Figure 5.40 shows an example of address translation.

**Figure 5.40** *Address translation*




## Translation Table

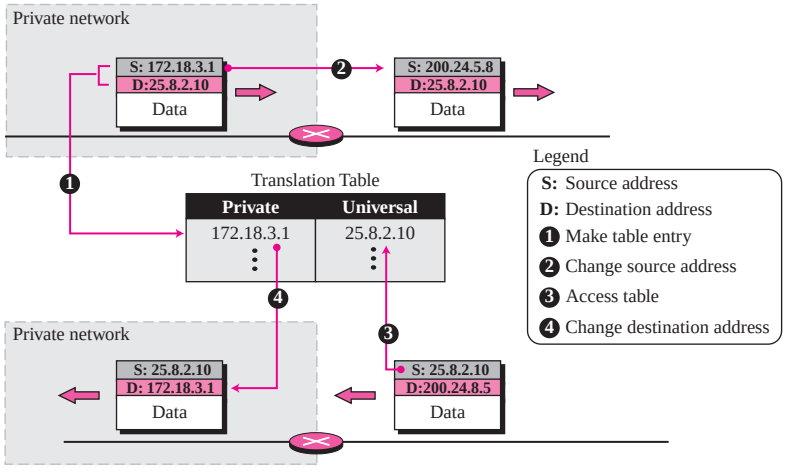
The reader may have noticed that translating the source addresses for an outgoing packet is straightforward. But how does the NAT router know the destination address for a packet coming from the Internet? There may be tens or hundreds of private IP

addresses, each belonging to one specific host. The problem is solved if the NAT router has a **translation table.**

### *Using One IP Address*

In its simplest form, a translation table has only two columns: the private address and the external address (destination address of the packet). When the router translates the source address of the outgoing packet, it also makes note of the destination address—where the packet is going. When the response comes back from the destination, the router uses the source address of the packet (as the external address) to find the private address of the packet. Figure 5.41 shows the idea.

**Figure 5.41** *Translation*




In this strategy, communication must always be initiated by the private network. The NAT mechanism described requires that the private network start the communication. As we will see, NAT is used mostly by ISPs that assign one single address to a customer. The customer, however, may be a member of a private network that has many private addresses. In this case, communication with the Internet is always initiated from the customer site, using a client program such as HTTP, TELNET, or FTP to access the corresponding server program. For example, when e-mail that originates from a non-customer site is received by the ISP e-mail server, it is stored in the mailbox of the customer until retrieved with a protocol such as POP.

A private network cannot run a server program for clients outside of its network if it is using NAT technology.

### *Using a Pool of IP Addresses*

Using only one global address by the NAT router allows only one private-network host to access the same external host. To remove this restriction, the NAT router can use a

pool of global addresses. For example, instead of using only one global address (200.24.5.8), the NAT router can use four addresses (200.24.5.8, 200.24.5.9, 200.24.5.10, and 200.24.5.11). In this case, four private-network hosts can communicate with the same external host at the same time because each pair of addresses defines a connection. However, there are still some drawbacks. No more than four connections can be made to the same destination. No private-network host can access two external server programs (e.g., HTTP and TELNET) at the same time. And, likewise, two private-network hosts cannot access the same external server program (e.g., HTTP or TELNET) at the same time.

#### *Using Both IP Addresses and Port Addresses*

To allow a many-to-many relationship between private-network hosts and external server programs, we need more information in the translation table. For example, suppose two hosts inside a private network with addresses 172.18.3.1 and 172.18.3.2 need to access the HTTP server on external host 25.8.3.2. If the translation table has five columns, instead of two, that include the source and destination port addresses and the transport layer protocol, the ambiguity is eliminated. Table 5.3 shows an example of such a table.

**Table 5.3** *Five-column translation table*

| *Private Address* | *Private Port* | *External Address* | *External Port* | *Transport Protocol* |
|---|---|---|---|---|
| 172.18.3.1 | 1400 | 25.8.3.2 | 80 | TCP |
| 172.18.3.2 | 1401 | 25.8.3.2 | 80 | TCP |
| … | … | … | … | … |

Note that when the response from HTTP comes back, the combination of source address (25.8.3.2) and destination port address (1400) defines the private network host to which the response should be directed. Note also that for this translation to work, the ephemeral port addresses (1400 and 1401) must be unique.

## 5.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], [Koz 05], [Ste 95].

### RFCs

Several RFCs deal with IPv4 addressing including RFC 917, RFC 927, RFC 930, RFC 932, RFC 940, RFC 950, RFC 1122, and RFC 1519.

## 5.7 KEY TERMS

address aggregation
address space
binary notation
block of addresses
class A address
class B address
class C address
class D address
class E address
classful addressing
classless addressing
classless interdomain routing (CIDR)
default mask
direct broadcast address
dotted-decimal notation
hexadecimal notation
hostid
IP address
limited broadcast address
loopback address
netid
network address
Network Address Translation (NAT)
network mask
prefix
prefix length
slash notation
subnet
subnet mask
subnetting
subnetwork
suffix
suffix length
supernet mask
supernetting
translation table

## 5.8 SUMMARY

- The identifier used in the IP layer of the TCP/IP protocol suite is called the Internet address or IP address. An IPv4 address is 32 bits long. An address space is the total number of addresses used by the protocol. The address space of IPv4 is $2^{32}$ or 4,294,967,296.
- In classful addressing, the IPv4 address space is divided into five classes: A, B, C, D, and E. An organization is granted a block in one of the three classes, A, B, or C. Classes D and E is reserved for special purposes. An IP address in classes A, B, and C is divided into netid and hostid.
- In classful addressing, the first address in the block is called the network address. It defines the network to which an address belongs. The network address is used in routing a packet to its destination network.
- A network mask or a default mask in classful addressing is a 32-bit number with $n$ leftmost bits all set to 1s and $(32 - n)$ rightmost bits all set to 0s. It is used by a router to find the network address from the destination address of a packet.
- The idea of splitting a network into smaller subnetworks is called subnetting. A subnetwork mask, like a network mask, is used to find the subnetwork address when a destination IP address is given. In supernetting, an organization can combine several class C blocks to create a larger range of addresses.
- In 1996, the Internet authorities announced a new architecture called classless addressing or CIDR that allows an organization to have a block of addresses of any size as long as the size of the block is a power of two.

- The address in classless addressing is also divided into two parts: the prefix and the suffix. The prefix plays the same role as the netid; the suffix plays the same role as the hostid. All addresses in the block have the same prefix; each address has a different suffix.
- Some of the blocks in IPv4 are reserved for special purposes. In addition, some addresses in a block are traditionally used for special addresses. These addresses are not assigned to any host.
- To improve the distribution of addresses, NAT technology has been created to allow the separation of private addresses in a network from the global addresses used in the Internet. A translation table can translate the private addresses, selected from the blocks allocated for this purpose, to global addresses. The translation table also translates the IP addresses as well as the port number for mapping from the private to global addresses and vice versa.

## 5.9 PRACTICE SET

### Exercises

1. What is the address space in each of the following systems?
   a. a system with 8-bit addresses
   b. a system with 16-bit addresses
   c. a system with 64-bit addresses
2. An address space has a total of 1,024 addresses. How many bits are needed to represent an address?
3. An address space uses three symbols: 0, 1, and 2 to represent addresses. If each address is made of 10 symbols, how many addresses are available in this system?
4. Change the following IP addresses from dotted-decimal notation to binary notation:
   a. 114.34.2.8
   b. 129.14.6.8
   c. 208.34.54.12
   d. 238.34.2.1
5. Change the following IP addresses from dotted-decimal notation to hexadecimal notation:
   a. 114.34.2.8
   b. 129.14.6.8
   c. 208.34.54.12
   d. 238.34.2.1
6. Change the following IP addresses from hexadecimal notation to binary notation:
   a. 0x1347FEAB
   b. 0xAB234102

**c.** 0x0123A2BE

**d.** 0x00001111

**7.** How many hexadecimal digits are needed to define the netid in each of the following classes?

**a.** Class A

**b.** Class B

**c.** Class C

**8.** Change the following IP addresses from binary notation to dotted-decimal notation:

**a.** `01111111 11110000 01100111 01111101`
**b.** `10101111 11000000 11111000 00011101`
**c.** `11011111 10110000 00011111 01011101`
**d.** `11101111 11110111 11000111 00011101`

**9.** Find the class of the following IP addresses:

**a.** 208.34.54.12

**b.** 238.34.2.1

**c.** 242.34.2.8

**d.** 129.14.6.8

**10.** Find the class of the following IP addresses:

**a.** `11110111 11110011 10000111 11011101`
**b.** `10101111 11000000 11110000 00011101`
**c.** `11011111 10110000 00011111 01011101`
**d.** `11101111 11110111 11000111 00011101`

**11.** Find the netid and the hostid of the following IP addresses:

**a.** 114.34.2.8

**b.** 132.56.8.6

**c.** 208.34.54.12

**d.** 251.34.98.5

**12.** Find the number of addresses in the range if the first address is 14.7.24.0 and the last address is 14.14.34.255.

**13.** If the first address in a range is 122.12.7.0 and there are 2048 addresses in the range, what is the last address?

**14.** Find the result of each operation:

**a.** NOT (22.14.70.34)

**b.** NOT (145.36.12.20)

**c.** NOT (200.7.2.0)

**d.** NOT (11.20.255.255)

**15.** Find the result of each operation:

**a.** (22.14.70.34) AND (255.255.0.0)

**b.** (12.11.60.12) AND (255.0.0.0)

**c.** (14.110.160.12) AND (255.200.140.0)

**d.** (28.14.40.100) AND (255.128.100.0)

**16.** Find the result of each operation:

**a.** (22.14.70.34) OR (255.255.0.0)

**b.** (12.11.60.12) OR (255.0.0.0)

**c.** (14.110.160.12) OR (255.200.140.0)

**d.** (28.14.40.100) OR (255.128.100.0)

**17.** In a class A subnet, we know the IP address of one of the hosts and the subnet mask as given below:

| IP Address: 25.34.12.56 | Subnet mask: 255.255.0.0 |
|---|---|

What is the first address (subnet address)? What is the last address?

**18.** In a class B subnet, we know the IP address of one of the hosts and the subnet mask as given below:

| IP Address: 131.134.112.66 | Subnet mask: 255.255.224.0 |
|---|---|

What is the first address (subnet address)? What is the last address?

**19.** In a class C subnet, we know the IP address of one of the hosts and the subnet mask as given below:

| IP Address: 202.44.82.16 | Subnet mask: 255.255.255.192 |
|---|---|

What is the first address (subnet address)? What is the last address?

**20.** Find the subnet mask in each case:

**a.** 1024 subnets in class A

**b.** 256 subnets in class B

**c.** 32 subnets in class C

**d.** 4 subnets in class C

**21.** In a block of addresses, we know the IP address of one host is 25.34.12.56/16. What is the first address (network address) and the last address (limited broadcast address) in this block?

**22.** In a block of addresses, we know the IP address of one host is 182.44.82.16/26. What is the first address (network address) and the last address (limited broadcast address) in this block?

**23.** In fixed-length subnetting, find the number of 1s that must be added to the mask if the number of desired subnets is _____.

**a.** 2

**b.** 62

**c.** 122

**d.** 250

**24.** An organization is granted the block 16.0.0.0/8. The administrator wants to create 500 fixed-length subnets.

**a.** Find the subnet mask.

**b.** Find the number of addresses in each subnet.

**c.** Find the first and the last address in the first subnet.

**d.** Find the first and the last address in the last subnet (subnet 500).

**25.** An organization is granted the block 130.56.0.0/16. The administrator wants to create 1024 subnets.

**a.** Find the subnet mask.

**b.** Find the number of addresses in each subnet.

**c.** Find the first and the last address in the first subnet.

**d.** Find the first and the last address in the last subnet (subnet 1024).

**26.** An organization is granted the block 211.17.180.0/24. The administrator wants to create 32 subnets.

**a.** Find the subnet mask.

**b.** Find the number of addresses in each subnet.

**c.** Find the first and the last address in the first subnet.

**d.** Find the first and the last address in the last subnet (subnet 32).

**27.** Write the following mask in slash notation (/*n*):

**a.** 255.255.255.0

**b.** 255.0.0.0

**c.** 255.255.224.0

**d.** 255.255.240.0

**28.** Find the range of addresses in the following blocks:

**a.** 123.56.77.32/29

**b.** 200.17.21.128/27

**c.** 17.34.16.0/23

**d.** 180.34.64.64/30

**29.** In classless addressing, we know the first and the last address in the block. Can we find the prefix length? If the answer is yes, show the process and give an example.

**30.** In classless addressing, we know the first address and the number of addresses in the block. Can we find the prefix length? If the answer is yes, show the process and give an example.

**31.** In classless addressing, can two blocks have the same prefix length? Explain.

**32.** In classless addressing, we know the first address and one of the addresses in the block (not necessarily the last address). Can we find the prefix length? Explain.

**33.** An ISP is granted a block of addresses starting with 150.80.0.0/16. The ISP wants to distribute these blocks to 2600 customers as follows:

**a.** The first group has 200 medium-size businesses; each needs approximately 128 addresses.

**b.** The second group has 400 small businesses; each needs approximately 16 addresses.

**c.** The third group has 2000 households; each needs 4 addresses.

Design the subblocks and give the slash notation for each subblock. Find out how many addresses are still available after these allocations.

**34.** An ISP is granted a block of addresses starting with 120.60.4.0/20. The ISP wants to distribute these blocks to 100 organizations with each organization receiving 8 addresses only. Design the subblocks and give the slash notation for each subblock. Find out how many addresses are still available after these allocations.

**35.** An ISP has a block of 1024 addresses. It needs to divide the addresses to 1024 customers. Does it need subnetting? Explain your answer.

CHAPTER

# 6

# *Delivery and Forwarding of IP Packets*

This chapter describes the delivery and forwarding of IP packets. **Delivery** refers to the way a packet is handled by the underlying networks under the control of the network layer. Concepts such as direct and indirect delivery are discussed. **Forwarding** refers to the way a packet is delivered to the next station. We discuss two trends in forwarding: forwarding based on destination address of the packet and forwarding based on the label attached to the packet.

## OBJECTIVES

*The chapter has several objectives:*

- To discuss the delivery of packets in the network layer and distinguish between direct and indirect delivery.
- To discuss the forwarding of packets in the network layer and distinguish between destination-address–based forwarding and label-based forwarding.
- To discuss different forwarding techniques, including next-hop, network-specific, host-specific, and default.
- To discuss the contents of routing tables in classful and classless addressing and some algorithms used to search the tables.
- To introduce MPLS technology and show how it can achieve label-based forwarding.
- To list the components of a router and explain the purpose of each component and their relations to other components.

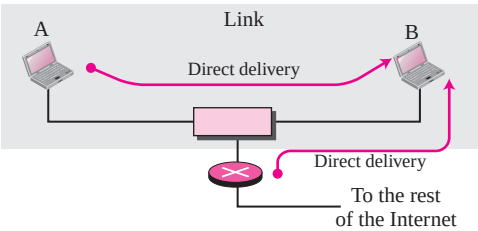
## 6.1 DELIVERY

The network layer supervises the handling of the packets by the underlying physical networks. We define this handling as the delivery of a packet. The delivery of a packet to its final destination is accomplished using two different methods of delivery: direct and indirect.

### Direct Delivery

In a **direct delivery,** the final destination of the packet is a host connected to the same physical network as the deliverer. Direct delivery occurs when the source and destination of the packet are located on the same physical network or if the delivery is between the last router and the destination host (see Figure 6.1).

**Figure 6.1** *Direct delivery*




The sender can easily determine if the delivery is direct. It can extract the network address of the destination (using the mask) and compare this address with the addresses of the networks to which it is connected. If a match is found, the delivery is direct.

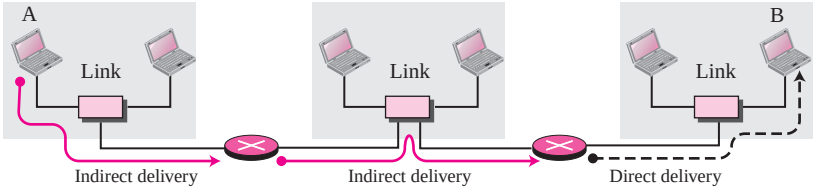
In direct delivery, the sender uses the destination IP address to find the destination physical address. The IP software then gives the destination IP address with the destination physical address to the data link layer for actual delivery. This process is called *mapping the IP address to the physical address*. Although this mapping can be done by finding a match in a table, we will see in Chapter 8 that a protocol called Address Resolution Protocol (ARP) dynamically maps an IP address to the corresponding physical address.

### Indirect Delivery

If the destination host is not on the same network as the deliverer, the packet is delivered indirectly. In an **indirect delivery,** the packet goes from router to router until it

reaches the one connected to the same physical network as its final destination. Figure 6.2 shows the concept of indirect delivery.

**Figure 6.2** *Indirect delivery*




In an indirect delivery, the sender uses the destination IP address and a routing table to find the IP address of the next router to which the packet should be delivered. The sender then uses ARP (see Chapter 8) to find the physical address of the next router. Note that in direct delivery, the address mapping is between the IP address of the final destination and the physical address of the final destination. In an indirect delivery, the address mapping is between the IP address of the next router and the physical address of the next router. Note that a delivery always involves one direct delivery but zero or more indirect deliveries. Note also that the last delivery is always a direct delivery.

## 6.2 FORWARDING

Forwarding means to place the packet in its route to its destination. Since the Internet today is made of a combination of links (networks), forwarding means to deliver the packet to the next hop (which can be the final destination or the intermediate connecting device). Although the IP protocol was originally designed as a connectionless protocol, today the tendency is to use IP as a connection-oriented protocol.

When IP is used as a connectionless protocol, forwarding is based on the destination address of the IP datagram; when the IP is used as a connection-oriented protocol, forwarding is based on the label attached to an IP datagram. We first discuss forwarding based on the destination address and then forwarding based on the label.

### Forwarding Based on Destination Address

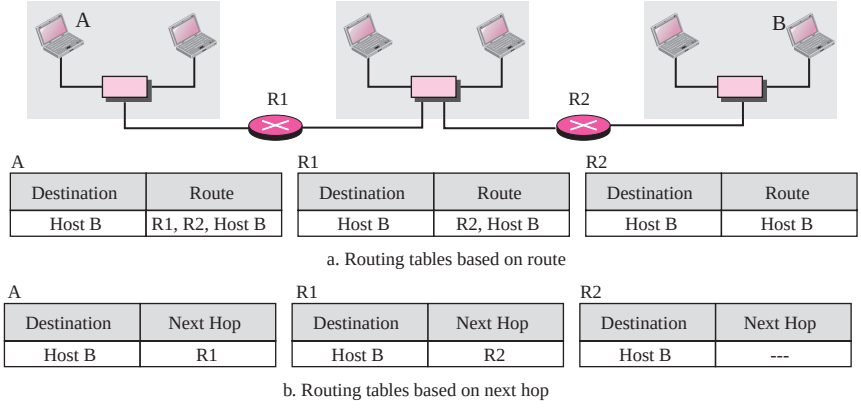
We first discuss forwarding based on the destination address. This is a traditional approach, which is prevalent today. In this case, forwarding requires a host or a router to have a routing table. When a host has a packet to send or when a router has received a packet to be forwarded, it looks at this table to find the route to the final destination. However, this simple solution is inefficient today in an internetwork such as the Internet because the number of entries needed in the routing table would make table lookups inefficient.

#### *Forwarding Techniques*

Several techniques can make the size of the routing table manageable and also handle issues such as security. We briefly discuss these methods here.

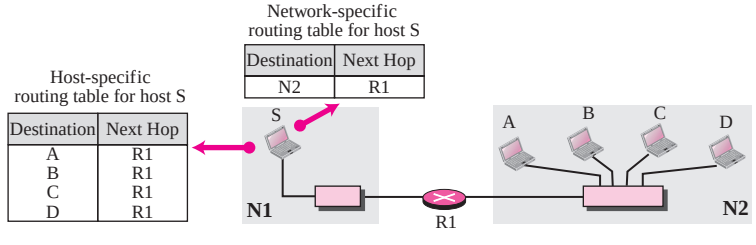
**Next-Hop Method** One technique to reduce the contents of a routing table is called the **next-hop method.** In this technique, the routing table holds only the address of the next hop instead of information about the complete route. The entries of a routing table must be consistent with each other. Figure 6.3 shows how routing tables can be simplified using this technique.

**Figure 6.3** *Next-hop method*




**Network-Specific Method** A second technique to reduce the routing table and simplify the searching process is called the **network-specific method.** Here, instead of having an entry for every destination host connected to the same physical network, we have only one entry that defines the address of the destination network itself. In other words, we treat all hosts connected to the same network as one single entity. For example, if 1000 hosts are attached to the same network, only one entry exists in the routing table instead of 1000. Figure 6.4 shows the concept.

**Figure 6.4** *Network-specific method*

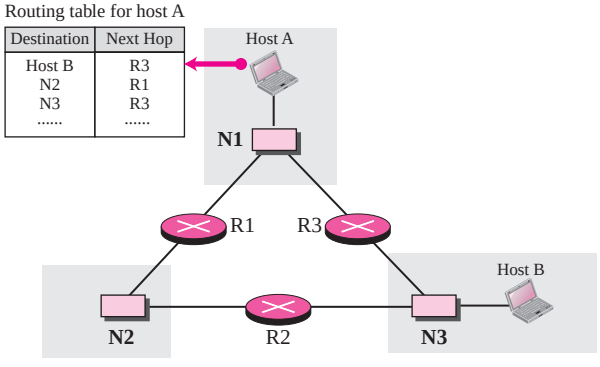



**Host-Specific Method** In the **host-specific method,** the destination host address is given in the routing table. The rationale behind this method is the inverse of the network-specific method. Here efficiency is sacrificed for other advantages: Although it is not efficient to put the host address in the routing table, there are occasions in

which the administrator wants to have more control over routing. For example, in Figure 6.5 if the administrator wants all packets arriving for host B delivered to router R3 instead of R1, one single entry in the routing table of host A can explicitly define the route.

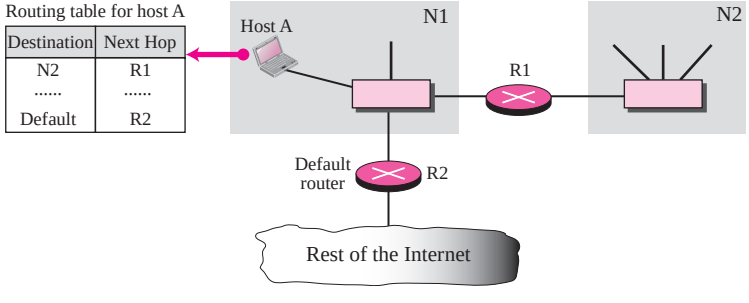
Host-specific routing is used for purposes such as checking the route or providing security measures.

**Figure 6.5** *Host-specific routing*




**Default Method** Another technique to simplify routing is called the **default method.** In Figure 6.6 host A is connected to a network with two routers. Router R1 routes the packets to hosts connected to network N2. However, for the rest of the Internet, router R2 is used. So instead of listing all networks in the entire Internet, host A can just have one entry called the *default* (normally defined as network address 0.0.0.0).

**Figure 6.6** *Default routing*




### *Forwarding with Classful Addressing*

As we mentioned in the previous chapter, classful addressing has several disadvantages. However, the existence of a default mask in a classful address makes the forwarding process simple. In this section, we first show the contents of a routing table

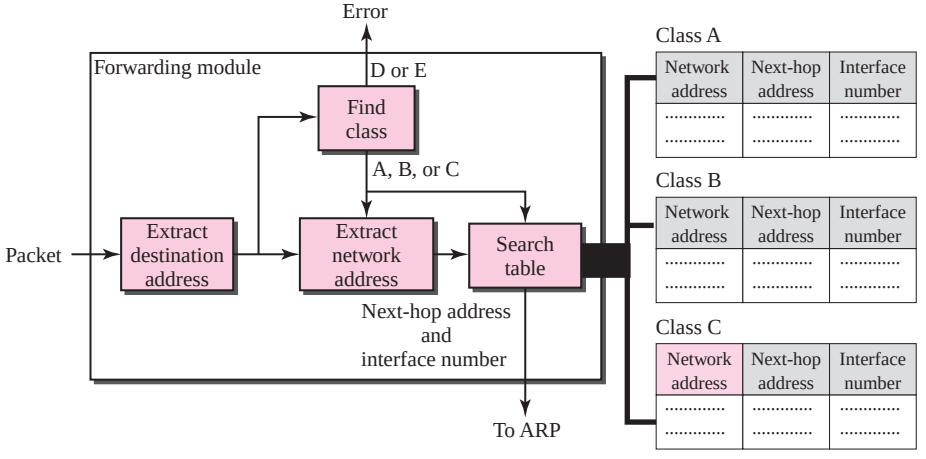
and forwarding module for the situation in which there is no subnetting. We then show how the module changes if subnetting is involved.

**Forwarding without Subnetting** In classful addressing, most of the routers in the global Internet are not involved in subnetting. Subnetting happens inside the organization. A typical forwarding module in this case can be designed using three tables, one for each unicast class (A, B, C). If the router supports multicasting, another table can be added to handle class D addresses. Having three different tables makes searching more efficient. Each routing table has a minimum of three columns:

1. The network address of the destination network tells us where the destination host is located. Note that we use network-specific forwarding and not the rarely used host-specific forwarding.
2. The next-hop address tells us to which router the packet must be delivered for an indirect delivery. This column is empty for a direct delivery.
3. The interface number defines the outgoing port from which the packet is sent out. A router is normally connected to several networks. Each connection has a different numbered port or interface. We show them as m0, m1, and so on.

Figure 6.7 shows a simplified module.

**Figure 6.7** *Simplified forwarding module in classful address without subnetting*




In its simplest form, the forwarding module follows these steps:

1. The destination address of the packet is extracted.
2. A copy of the destination address is used to find the class of the address. This is done by shifting the copy of the address 28 bits to the right. The result is a 4-bit number between 0 and 15. If the result is
   a. 0 to 7, the class is A.
   b. 8 to 11, the class is B.
   c. 12 or 13, the class is C

**d.** 14, the class is D.

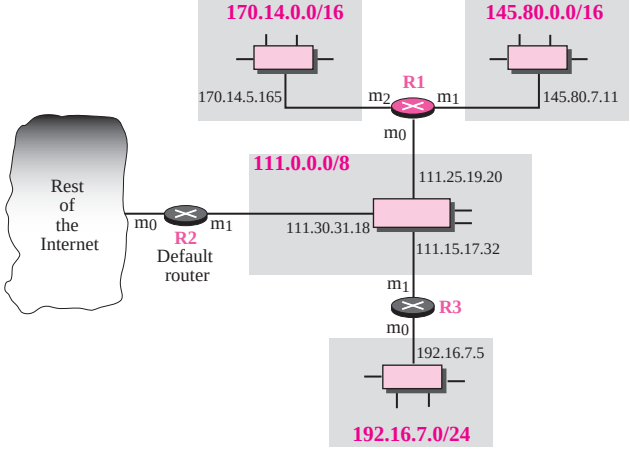
**e.** 15, the class is E.

3. The result of Step 2 for class A, B, or C and the destination address are used to extract the network address. This is done by masking off (changing to 0s) the rightmost 8, 16, or 24 bits based on the class.
4. The class of the address and the network address are used to find next-hop information. The class determines which table is to be searched. The module searches this table for the network address. If a match is found, the **next-hop address** and the interface number of the output port are extracted from the table. If no match is found, the default is used.
5. The ARP module (Chapter 8) uses the next-hop address and the interface number to find the physical address of the next router. It then asks the data link layer to deliver the packet to the next hop.

### Example 6.1

Figure 6.8 shows an imaginary part of the Internet. Show the routing tables for router R1.

**Figure 6.8** *Configuration for routing, Example 1*




### Solution

Figure 6.9 shows the three tables used by router R1. Note that some entries in the next-hop address column are empty because in these cases, the destination is in the same network to which the router is connected (direct delivery). In these cases, the next-hop address used by ARP is simply the destination address of the packet as we will see in Chapter 8.

### Example 6.2

Router R1 in Figure 6.8 receives a packet with destination address 192.16.7.14. Show how the packet is forwarded.

**Figure 6.9** *Tables for Example 6.1*

Class A

| Network address | Next-hop address | Interface |
|---|---|---|
| 111.0.0.0 | ---------- | m0 |

Class B

| Network address | Next-hop address | Interface |
|---|---|---|
| 145.80.0.0 | ---------- | m1 |
| 170.14.0.0 | ---------- | m2 |

Class C

| Network address | Next-hop address | Interface |
|---|---|---|
| 192.16.7.0 | 111.15.17.32 | m0 |

Default: 111.30.31.18, m0

**Solution**

The destination address in binary is 11000000 00010000 00000111 00001110. A copy of the address is shifted 28 bits to the right. The result is 00000000 00000000 00000000 0000**1100** or 12. The destination network is class C. The network address is extracted by masking off the leftmost 24 bits of the destination address; the result is 192.16.7.0. The table for Class C is searched. The network address is found in the first row. The next-hop address 111.15.17.32. and the interface m0 are passed to ARP (see Chapter 8).

### Example 6.3

Router R1 in Figure 6.8 receives a packet with destination address 167.24.160.5. Show how the packet is forwarded.

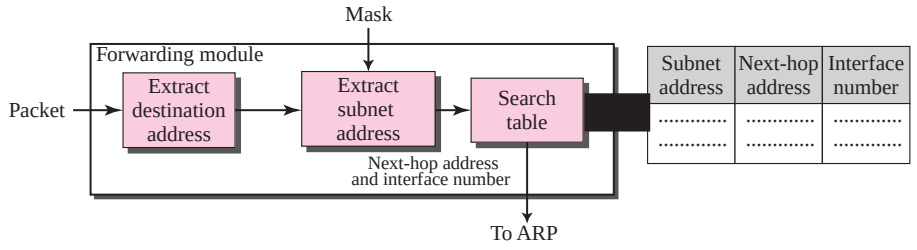
**Solution**

The destination address in binary is 10100111 00011000 10100000 00000101. A copy of the address is shifted 28 bits to the right. The result is 00000000 00000000 00000000 0000**1010** or 10. The class is B. The network address can be found by masking off 16 bits of the destination address, the result is 167.24.0.0. The table for Class B is searched. No matching network address is found. The packet needs to be forwarded to the default router (the network is somewhere else in the Internet). The next-hop address 111.30.31.18 and the interface number m0 are passed to ARP.

**Forwarding with Subnetting** In classful addressing, subnetting happens inside the organization. The routers that handle subnetting are either at the border of the organization site or inside the site boundary. If the organization is using variable-length subnetting, we need several tables; otherwise, we need only one table. Figure 6.10 shows a simplified module for fixed-length subnetting.

1. The module extracts the destination address of the packet.
2. If the destination address matches any of the host-specific addresses in the table, the next-hop and the interface number is extracted from the table.
3. The destination address and the mask are used to extract the subnet address.
4. The table is searched using the subnet address to find the next-hop address and the interface number. If no match is found, the default is used.
5. The next-hop address and the interface number are given to ARP (see Chapter 8).

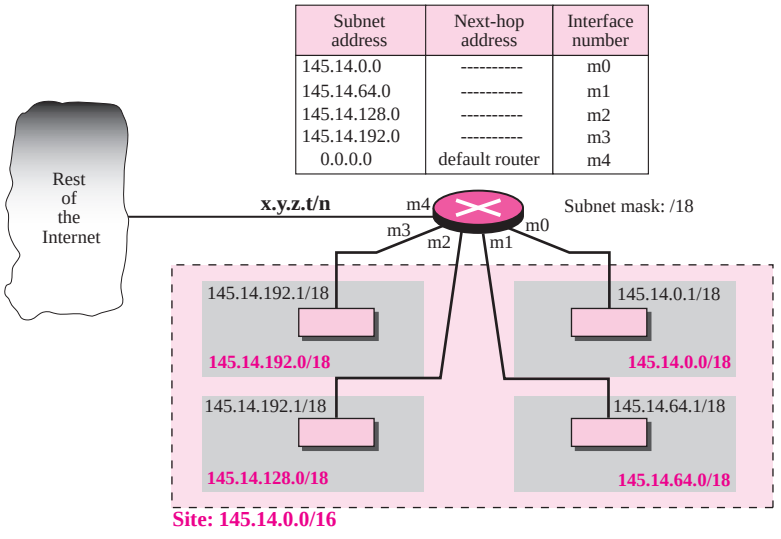
**Figure 6.10** *Simplified forwarding module in classful address with subnetting*




## Example 6.4

Figure 6.11 shows a router connected to four subnets.

**Figure 6.11** *Configuration for Example 6.4*




Note several points. First, the site address is 145.14.0.0/16 (a class B address). Every packet with destination address in the range 145.14.0.0 to 145.14.255.255 is delivered to the interface m4 and distributed to the final destination subnet by the router. Second, we have used the address x.y.z.t/*n* for the interface m4 because we do not know to which network this router is connected. Third, the table has a default entry for packets that are to be sent out of the site. The router is configured to apply the subnet mask /18 to any destination address.

## Example 6.5

The router in Figure 6.11 receives a packet with destination address 145.14.32.78. Show how the packet is forwarded.

**Solution**

The mask is /18. After applying the mask, the subnet address is 145.14.0.0. The packet is delivered to ARP (see Chapter 8) with the next-hop address 145.14.32.78 and the outgoing interface m0.

### Example 6.6

A host in network 145.14.0.0 in Figure 6.11 has a packet to send to the host with address 7.22.67.91. Show how the packet is routed.

**Solution**

The router receives the packet and applies the mask (/18). The network address is 7.22.64.0. The table is searched and the address is not found. The router uses the address of the default router (not shown in figure) and sends the packet to that router.

### *Forwarding with Classless Addressing*

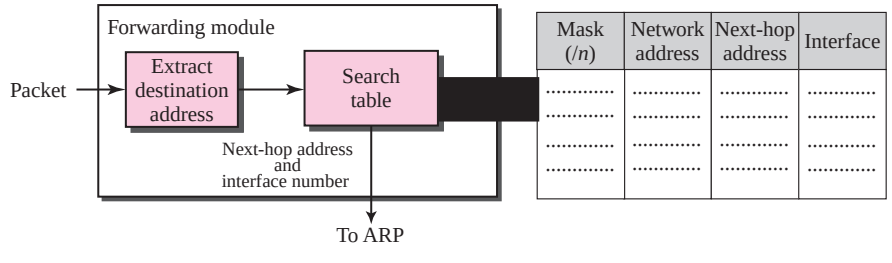
In classless addressing, the whole address space is one entity; there are no classes. This means that forwarding requires one row of information for each block involved. The table needs to be searched based on the network address (first address in the block). Unfortunately, the destination address in the packet gives no clue about the network address (as it does in classful addressing).

To solve the problem, we need to include the mask (/*n*) in the table; we need to have an extra column that includes the mask for the corresponding block. In other words, although a classful routing table can be designed with three columns, a classless routing table needs at least four columns.

**In classful addressing we can have a routing table with three columns; in classless addressing, we need at least four columns.**

Figure 6.12 shows a simple forwarding module for classless addressing. Note that network address extraction is done at the same time as table searching because there is no inherent information in the destination address that can be used for network address extraction.

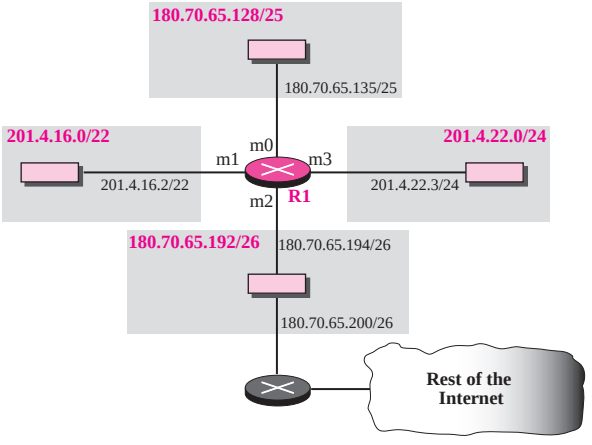
**Figure 6.12** *Simplified forwarding module in classless address*




### Example 6.7

Make a routing table for router R1 using the configuration in Figure 6.13.

**Figure 6.13** *Configuration for Example 6.7*




### Solution

Table 6.1 shows the corresponding table.

**Table 6.1** *Routing table for router R1 in Figure 6.13*

| *Mask* | *Network Address* | *Next Hop* | *Interface* |
|---|---|---|---|
| /26 | 180.70.65.192 | - | m2 |
| /25 | 180.70.65.128 | - | m0 |
| /24 | 201.4.22.0 | - | m3 |
| /22 | 201.4.16.0 | .... | m1 |
| Default | Default | 180.70.65.200 | m2 |

## Example 6.8

Show the forwarding process if a packet arrives at R1 in Figure 6.13 with the destination address 180.70.65.140.

### Solution

The router performs the following steps:

1. The first mask (/26) is applied to the destination address. The result is 180.70.65.128, which does not match the corresponding network address.
2. The second mask (/25) is applied to the destination address. The result is 180.70.65.128, which matches the corresponding network address. The next-hop address (the destination address of the packet in this case) and the interface number m0 are passed to ARP (see Chapter 8) for further processing.

## Example 6.9

Show the forwarding process if a packet arrives at R1 in Figure 6.13 with the destination address 201.4.22.35.

**Solution**

The router performs the following steps:

1. The first mask (/26) is applied to the destination address. The result is 201.4.22.0, which does not match the corresponding network address (row 1).
2. The second mask (/25) is applied to the destination address. The result is 201.4.22.0, which does not match the corresponding network address (row 2).
3. The third mask (/24) is applied to the destination address. The result is 201.4.22.0, which matches the corresponding network address. The destination address of the packet and the interface number m3 are passed to ARP (see Chapter 8).

### Example 6.10

Show the forwarding process if a packet arrives at R1 in Figure 6.13 with the destination address 18.24.32.78.

**Solution**

This time all masks are applied to the destination address, but no matching network address is found. When it reaches the end of the table, the module gives the next-hop address 180.70.65.200 and interface number m2 to ARP (see Chapter 8). This is probably an outgoing package that needs to be sent, via the default router, to someplace else in the Internet.

### Example 6.11

Now let us give a different type of example. Can we find the configuration of a router if we know only its routing table? The routing table for router R1 is given in Table 6.2. Can we draw its topology?

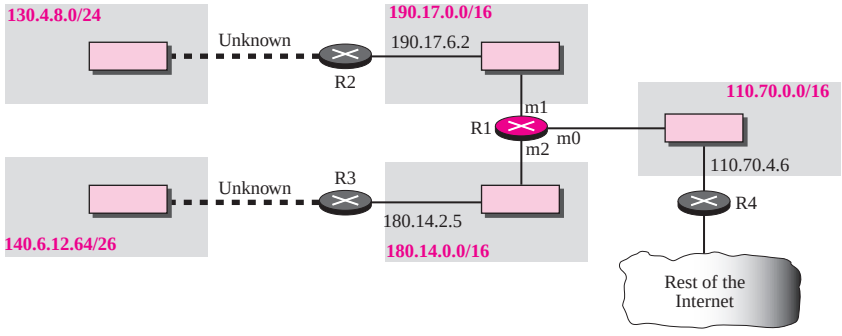
**Table 6.2** *Routing table for Example 6.11*

| *Mask* | *Network Address* | *Next-Hop Address* | *Interface Number* |
|---|---|---|---|
| /26 | 140.6.12.64 | 180.14.2.5 | m2 |
| /24 | 130.4.8.0 | 190.17.6.2 | m1 |
| /16 | 110.70.0.0 | ---------- | m0 |
| /16 | 180.14.0.0 | ---------- | m2 |
| /16 | 190.17.0.0 | ---------- | m1 |
| Default | Default | 110.70.4.6 | m0 |

**Solution**

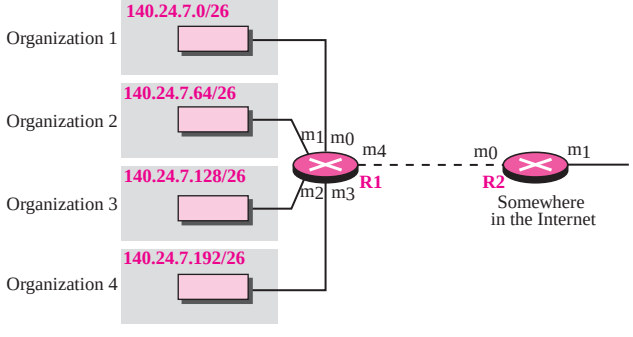
We know some facts but we don't have all for a definite topology. We know that router R1 has three interfaces: m0, m1, and m2. We know that there are three networks directly connected to router R1. We know that there are two networks indirectly connected to R1. There must be at least three other routers involved (see next-hop column). We know to which networks these routers are connected by looking at their IP addresses. So we can put them at their appropriate place. We know that one router, the default router, is connected to the rest of the Internet. But there is some missing information.We do not know if network 130.4.8.0 is directly connected to router R2 or through a point-to-point network (WAN) and another router. We do not know if network 140.6.12.64 is connected to router R3 directly or through a point-to-point network (WAN) and another router. Point-to-point networks normally do not have an entry in the routing table because no hosts are connected to them. Figure 6.14 shows our guessed topology.

**Figure 6.14** *Guessed topology for Example 6.11*




**Address Aggregation** When we use classful addressing, there is only one entry in the routing table for each site outside the organization. The entry defines the site even if that site is subnetted. When a packet arrives at the router, the router checks the corresponding entry and forwards the packet accordingly. When we use classless addressing, it is likely that the number of routing table entries will increase. This is because the intent of classless addressing is to divide up the whole address space into manageable blocks. The increased size of the table results in an increase in the amount of time needed to search the table. To alleviate the problem, the idea of **address aggregation** was designed. In Figure 6.15 we have two routers.

**Figure 6.15** *Address aggregation*




| Mask | Network address | Next-hop address | Interface |
|---|---|---|---|
| /26 | 140.24.7.0 | ---------- | m0 |
| /26 | 140.24.7.64 | ---------- | m1 |
| /26 | 140.24.7.128 | ---------- | m2 |
| /26 | 140.24.7.192 | ---------- | m3 |
| /0 | 0.0.0.0 | default router | m4 |

Routing table for R1

| Mask | Network address | Next-hop address | Interface |
|---|---|---|---|
| /24 | 140.24.7.0 | ---------- | m0 |
| /0 | 0.0.0.0 | default router | m1 |

Routing table for R2

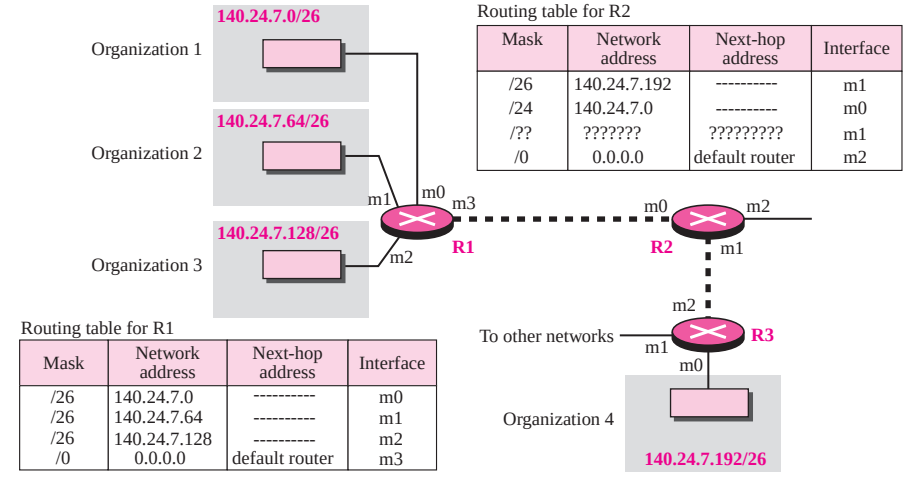
R1 is connected to networks of four organizations that each use 64 addresses. R2 is somewhere far from R1. R1 has a longer routing table because each packet must be correctly routed to the appropriate organization. R2, on the other hand, can have a very

small routing table. For R2, any packet with destination 140.24.7.0 to 140.24.7.255 is sent out from interface m0 regardless of the organization number. This is called address aggregation because the blocks of addresses for four organizations are aggregated into one larger block. R2 would have a longer routing table if each organization had addresses that could not be aggregated into one block.

Note that although the idea of address aggregation is similar to the idea of subnetting, we do not have a common site here; the network for each organization is independent. In addition, we can have several levels of aggregation.

**Longest Mask Matching** What happens if one of the organizations in the previous figure is not geographically close to the other three? For example, if organization 4 cannot be connected to router R1 for some reason, can we still use the idea of address aggregation and still assign block 140.24.7.192/26 to organization 4? The answer is yes because routing in classless addressing uses another principle, **longest mask matching.** This principle states that the routing table is sorted from the longest mask to the shortest mask. In other words, if there are three masks, /27, /26, and /24, the mask /27 must be the first entry and /24 must be last. Let us see if this principle solves the situation in which organization 4 is separated from the other three organizations. Figure 6.16 shows the situation.

**Figure 6.16** *Longest mask matching*




Routing table for R2

| Mask | Network address | Next-hop address | Interface |
|---|---|---|---|
| /26 | 140.24.7.192 | ---------- | m1 |
| /24 | 140.24.7.0 | ---------- | m0 |
| /?? | ??????? | ????????? | m1 |
| /0 | 0.0.0.0 | default router | m2 |

Routing table for R1

| Mask | Network address | Next-hop address | Interface |
|---|---|---|---|
| /26 | 140.24.7.0 | ---------- | m0 |
| /26 | 140.24.7.64 | ---------- | m1 |
| /26 | 140.24.7.128 | ---------- | m2 |
| /0 | 0.0.0.0 | default router | m3 |

Routing table for R3

| Mask | Network address | Next-hop address | Interface |
|---|---|---|---|
| /26 | 140.24.7.192 | ---------- | m0 |
| /?? | ??????? | ????????? | m1 |
| /0 | 0.0.0.0 | default router | m2 |

Suppose a packet arrives for organization 4 with destination address 140.24.7.200. The first mask at router R2 is applied, which gives the network address 140.24.7.192. The packet is routed correctly from interface m1 and reaches organization 4. If, however, the routing table was not stored with the longest prefix first, applying the /24 mask would result in the incorrect routing of the packet to router R1.

**Hierarchical Routing** To solve the problem of gigantic routing tables, we can create a sense of hierarchy in the routing tables. In Chapter 1, we mentioned that the Internet today has a sense of hierarchy. We said that the Internet is divided into backbone, regional and local ISPs. If the routing table has a sense of hierarchy like the Internet architecture, the routing table can decrease in size.

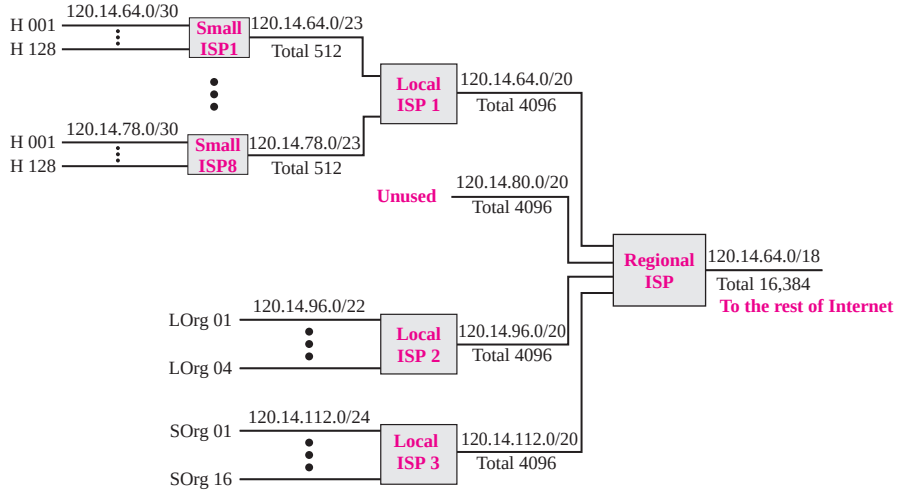
Let us take the case of a local ISP. A local ISP can be assigned a single, but large, block of addresses with a certain prefix length. The local ISP can divide this block into smaller blocks of different sizes, and assign these to individual users and organizations, both large and small. If the block assigned to the local ISP starts with a.b.c.d/*n*, the ISP can create blocks starting with e.f.g.h/*m*, where *m* may vary for each customer and is greater than *n*.

How does this reduce the size of the routing table? The rest of the Internet does not have to be aware of this division. All customers of the local ISP are defined as a.b.c.d/*n* to the rest of the Internet. Every packet destined for one of the addresses in this large block is routed to the local ISP. There is only one entry in every router in the world for all of these customers. They all belong to the same group. Of course, inside the local ISP, the router must recognize the subblocks and route the packet to the destined customer. If one of the customers is a large organization, it also can create another level of hierarchy by subnetting and dividing its subblock into smaller subblocks (or sub-subblocks). In classless routing, the levels of hierarchy are unlimited so long as we follow the rules of classless addressing.

### Example 6.12

As an example of hierarchical routing, let us consider Figure 6.17. A regional ISP is granted 16,384 addresses starting from 120.14.64.0. The regional ISP has decided to divide this block into 4 subblocks, each with 4096 addresses. Three of these subblocks are assigned to three local

**Figure 6.17** *Hierarchical routing with ISPs*



ISPs, the second subblock is reserved for future use. Note that the mask for each block is /20 because the original block with mask /18 is divided into 4 blocks.

The first local ISP has divided its assigned subblock into 8 smaller blocks and assigned each to a small ISP. Each small ISP provides services to 128 households (H001 to H128), each using four addresses. Note that the mask for each small ISP is now /23 because the block is further divided into 8 blocks. Each household has a mask of /30, because a household has only 4 addresses ($2^{32-30} = 4$). The second local ISP has divided its block into 4 blocks and has assigned the addresses to 4 large organizations (LOrg01 to LOrg04). Note that each large organization has 1024 addresses and the mask is /22.

The third local ISP has divided its block into 16 blocks and assigned each block to a small organization (SOrg01 to SOrg15). Each small organization has 256 addresses and the mask is /24. There is a sense of hierarchy in this configuration. All routers in the Internet send a packet with destination address 120.14.64.0 to 120.14.127.255 to the regional ISP. The regional ISP sends every packet with destination address 120.14.64.0 to 120.14.79.255 to Local ISP1. Local ISP1 sends every packet with destination address 120.14.64.0 to 120.14.64.3 to H001.

**Geographical Routing** To decrease the size of the routing table even further, we need to extend hierarchical routing to include geographical routing. We must divide the entire address space into a few large blocks. We assign a block to America, a block to Europe, a block to Asia, a block to Africa, and so on. The routers of ISPs outside of Europe will have only one entry for packets to Europe in their routing tables. The routers of ISPs outside of America will have only one entry for packets to North America in their routing tables. And so on.

### *Routing Table Search Algorithms*

The algorithms in classful addressing that search the routing tables must be changed to make classless routing more efficient. This includes the algorithms that update routing tables. We will discuss this updating issue in Chapter 11.

**Searching in Classful Addressing** In classful addressing, the routing table is organized as a list. However, to make searching more efficient, the routing table can be divided into three tables (sometimes called buckets), one for each class. When the packet arrives, the router applies the default mask (which is inherent in the address itself) to find the corresponding bucket (A, B, or C). The router then searches the corresponding bucket instead of the whole table. Some routers even assign 8 buckets for class A, 4 buckets for class B, and 2 buckets for class C based on the outcome of the class finding process.

**Searching in Classless Addressing** In classless addressing, there is no network information in the destination address. The simplest, but not the most efficient, search method is called the **longest prefix match** (as we discussed before**)**. The routing table can be divided into buckets, one for each prefix. The router first tries the longest prefix. If the destination address is found in this bucket, the search is complete. If the address is not found, the next prefix is searched. And so on. It is obvious that this type of search takes a long time.

One solution is to change the data structure used for searching. Instead of a list, other data structures (such as a tree or a binary tree) can be used. One candidate is a trie (a special kind of tree). However, this discussion is beyond the scope of this book.

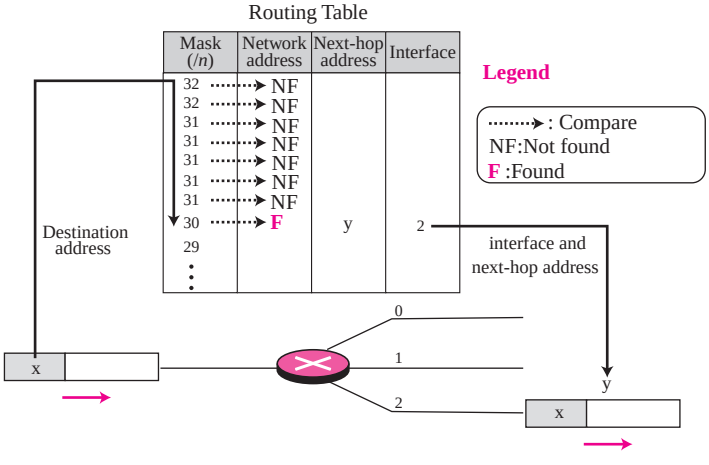
## Forwarding Based on Label

In 1980s, an effort started to somehow change IP to behave like a connection-oriented protocol in which the routing is replaced by switching. As we discussed in Chapter 4, in a connectionless network (datagram approach), a router forwards a packet based on the destination address in the header of packet. On the other hand, in a connection-oriented network (virtual-circuit approach), a switch forwards a packet based on the label *attached* to a packet. Routing is normally based on searching the contents of a table; switching can be done by accessing a table using an index. In other words, routing involves searching; switching involves accessing.

### Example 6.13

Figure 6.18 shows a simple example of searching in a routing table using the longest match algorithm. Although there are some more efficient algorithms today, the principle is the same.

**Figure 6.18** *Example 6.13: Forwarding based on destination address*




When the forwarding algorithm gets the destination address of the packet, it needs to delve into the mask column. For each entry, it needs to apply the mask to find the destination network address. It then needs to check the network addresses in the table until it finds the match. The router then extracts the next-hop address and the interface number to be delivered to the ARP protocol for delivery of the packet to the next hop.

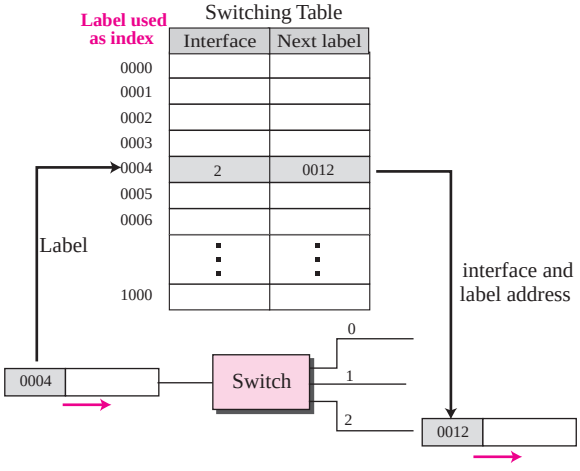
### Example 6.14

Figure 6.19 shows a simple example of using a label to access a switching table. Since the labels are used as the index to the table, finding the information in the table is immediate.

### *MPLS*

During the 1980s, several vendors created routers that implement switching technology. Later IETF approved a standard that is called Multi-Protocol Label Switching. In

**Figure 6.19** *Example 6.14: Forwarding based on label*

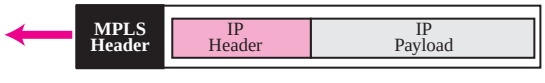



this standard, some conventional routers in the Internet can be replaced by MPLS routers that can behave like a router and a switch. When behaving like a router, MPLS can forward the packet based on the destination address; when behaving like a switch, it can forward a packet based on the label.

### *A New Header*

To simulate connection-oriented switching using a protocol like IP, the first thing that is needed is to add a field to the packet that carry the label discussed in Chapter 4. The IPv4 packet format does not allow this extension (although this field is provided in IPv6 packet format, as we will see in Chapter 27). The solution is to encapsulate the IPv4 packet in an MPLS packet (as though MPLS is a layer between the data link layer and the network layer). The whole IP packet is encapsulated as the payload in an MPLS packet and MPLS header is added. Figure 6.20 shows the encapsulation.

**Figure 6.20** *MPLS header added to an IP packet*

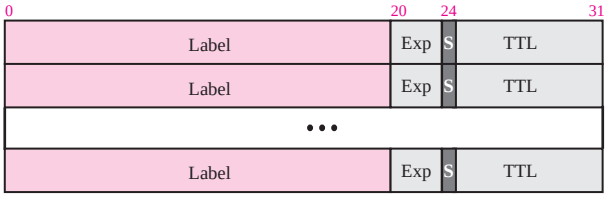



The MPLS header is actually a stack of subheaders that is used for multilevel hierarchical switching as we discuss shortly. Figure 6.21 shows the format of an MPLS header in which each subheader is 32 bits (4 bytes) long.

The following is a brief description of each field:

- **Label.** This 20-bit field defines the label that is used to index the routing table in the router.
- **Exp.** This 3-bit field is reserved for experimental purposes.

**Figure 6.21** *MPLS header made of stack of labels*




- **S.** The one-bit stack field defines the situation of the subheader in the stack. When the bit is 1, it means that the header is the last one in the stack.
- **TTL.** This 8-bit field is similar to the TTL field in the IP datagram (see Chapter 7). Each visited router decrement the value of this field. When it reaches zero, the packet is discarded to prevent looping.

### *Hierarchical Switching*

A stack of labels in MPLS allows hierarchical switching. This is similar to conventional hierarchical routing. For example, a packet with two labels can use the top label to forward the packet through switches outside an organization; the bottom label can be used to route the packet inside the organization to reach the destination subnet.

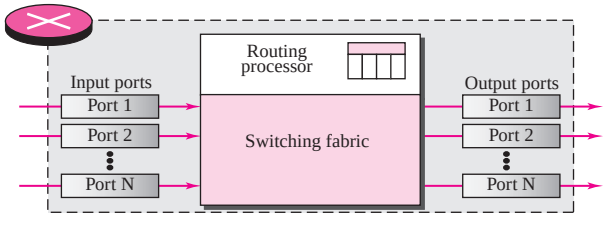
## 6.3 STRUCTURE OF A ROUTER

In our discussion of forwarding and routing, we represented a router as a black box that accepts incoming packets from one of the input ports (interfaces), uses a routing table to find the output port from which the packet departs, and sends the packet from this output port. In this section we open the black box and look inside. However, our discussion won't be very detailed; entire books have been written about routers. We just give an overview to the reader.

### Components

We can say that a router has four components: **input ports, output ports,** the **routing processor,** and the **switching fabric,** as shown in Figure 6.22.

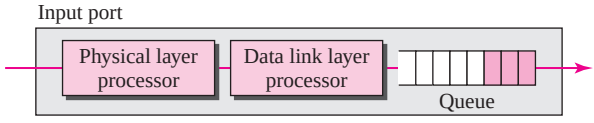
**Figure 6.22** *Router components*



### Input Ports

Figure 6.23 shows a schematic diagram of an input port.

**Figure 6.23** *Input port*

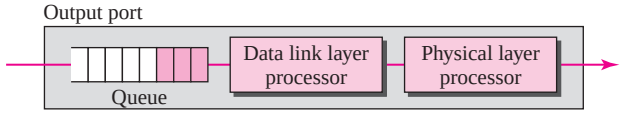



An input port performs the physical and data link layer functions of the router. The bits are constructed from the received signal. The packet is decapsulated from the frame. Errors are detected and corrected. The packet is ready to be forwarded by the network layer. In addition to a physical layer processor and a data link processor, the input port has buffers (queues) to hold the packets before they are directed to the switching fabric.

### Output Ports

An output port performs the same functions as the input port, but in the reverse order. First the outgoing packets are queued, then the packet is encapsulated in a frame, and finally the physical layer functions are applied to the frame to create the signal to be sent on the line. Figure 6.24 shows a schematic diagram of an output port.

**Figure 6.24** *Output port*




### Routing Processor

The routing processor performs the functions of the network layer. The destination address is used to find the address of the next hop and, at the same time, the output port number from which the packet is sent out. This activity is sometimes referred to as *table lookup* because the routing processor searches the routing table. In the newer routers, this function of the routing processor is being moved to the input ports to facilitate and expedite the process.

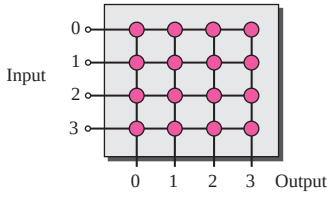
### Switching Fabrics

The most difficult task in a router is to move the packet from the input queue to the output queue. The speed with which this is done affects the size of the input/output queue and the overall delay in packet delivery. In the past, when a router was actually a dedicated computer, the memory of the computer or a bus was used as the switching fabric. The input port stored the packet in memory; the output port got the packet from the

memory. Today, routers use a variety of switching fabrics. We briefly discuss some of these fabrics here.

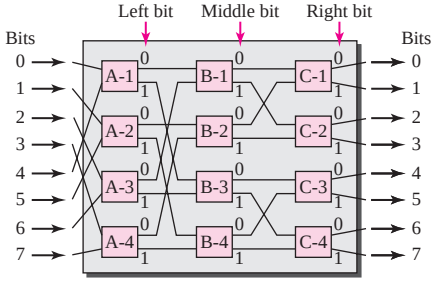
**Crossbar Switch** The simplest type of switching fabric is the crossbar switch shown in Figure 6.25. A **crossbar switch** connects *n* inputs to *n* outputs in a grid, using electronic microswitches at each **crosspoint.**

**Figure 6.25** *Crossbar switch*



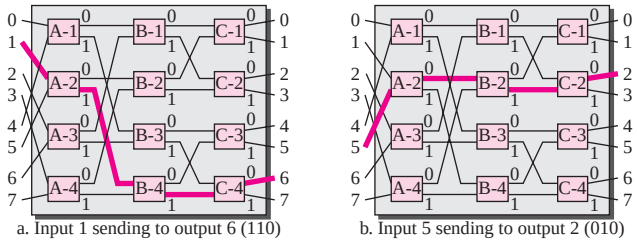

**Banyan Switch** More realistic than the crossbar switch is the **banyan switch** (named after the banyan tree) as shown in Figure 6.26.

**Figure 6.26** *A banyan switch*




A banyan switch is a multistage switch with microswitches at each stage that route the packets based on the output port represented as a binary string. For *n* inputs and *n* outputs, we have $\log_2(n)$ stages with $n/2$ microswitches at each stage. The first stage routes the packet based on the highest order bit of the binary string. The second stage routes the packets based on the second highest order bit, and so on. Figure 6.27 shows a banyan switch with eight inputs and eight outputs. The number of stages is $\log_2(8) = 3$. Suppose a packet has arrived at input port 1 and must go to output port 6 (110 in binary). The first microswitch (A-2) routes the packet based on the first bit (1), the second microswitch (B-4) routes the packet based on the second bit (1), and the third microswitch (C-4) routes the packet based on the third bit (0). In part b, a packet has arrived at input port 5 and must go to output port 2 (010 in binary). The first microswitch (A-2) routes the packet based on the first bit (0), the second microswitch (B-2) routes the packet based on the second bit (1), and the third microswitch (C-2) routes the packet based on the third bit (0).

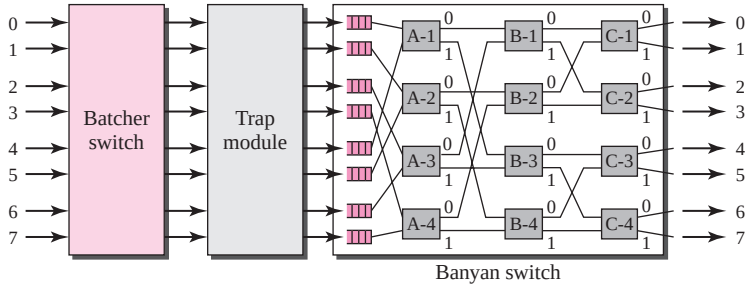
**Figure 6.27** *Examples of routing in a banyan switch*




a. Input 1 sending to output 6 (110) b. Input 5 sending to output 2 (010)

**Batcher-Banyan Switch** The problem with the banyan switch is the possibility of internal collision even when two packets are not heading for the same output port. We can solve this problem by sorting the arriving packets based on their destination port. K. E. Batcher designed a switch that comes before the banyan switch and sorts the incoming packets according to their final destination. The combination is called the **Batcher-banyan switch** (see Figure 6.28).

**Figure 6.28** *Batcher-banyan switch*




The sorting switch uses hardware merging techniques, but we will not discuss the details here. Normally, another hardware module called a trap is added between the Batcher switch and the banyan switch. The trap module prevents duplicate packets (packets with the same output destination) from passing to the banyan switch simultaneously. Only one packet for each destination is allowed at each tick; if there is more than one, they wait for the next tick.

## 6.4 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], and [Kur & Ros 08].

### RFCs

Forwarding is discussed in RFC 1812, RFC 1971, and RFC 1980. MPLS is discussed in RFC 3031, RFC 3032, RFC 3036, and RFC 3212.

## 6.5 KEY TERMS

banyan switch
Batcher-banyan switch
crossbar switch
crosspoint
default method
delivery
direct delivery
forwarding
host-specific method
indirect delivery
input ports
longest mask matching
longest prefix match
network-specific method
next-hop address
next-hop method
output ports
routing processor

## 6.6 SUMMARY

- The network layer supervises the handling of the packets by the underlying physical networks.We define this handling as the delivery of a packet. The delivery of a packet is called direct if the deliverer (host or router) and the destination are on the same network. The delivery of a packet is called indirect if the deliverer (host or router) and the destination are on different networks.
- Forwarding means to deliver the packet to the next hop. Two categories of forwarding were discussed in this chapter: forwarding based on the destination address of the IP datagram and forwarding based on the label attached to an IP datagram. The first searches a routing table to forward a packet; the second uses the label as an index to a switching table to forward a packet.
- We discussed several methods in destination-address–based forwarding including host-specific method, next-hop method, network-specific method, and the default method.
- In destination-address–based forwarding, the routing table for classful forwarding can have three columns. The routing table for classless addressing needs at least four columns. Address aggregation simplifies the forwarding process in classless addressing. Longest mask matching is required in classless addressing.

- In label-based forwarding, a switching table is used instead of a routing table. The Multi-Protocol Label Switching (MPLS) is the standard approved by IETF, which adds a pseudo layer to the TCP/IP protocol suite by encapsulating the IP packet in an MPLS packet.
- A router is normally made of four components: input ports, output ports, the routing processor, and the switching fabric.

## 6.7 PRACTICE SET

### Exercises

1. A host with IP address 137.23.56.23/16 sends a packet to a host with IP address 137.23.67.9/16. Is the delivery direct or indirect? Assume no subnetting.
2. A host with IP address 137.23.56.23/16 sends a packet to a host with IP address 142.3.6.9/24. Is the delivery direct or indirect? Assume no subnetting.
3. In Figure 6.8, find the routing table for router R2.
4. In Figure 6.8, find the routing table for router R3.
5. A packet arrives at router R1 in Figure 6.8 with destination address 192.16.7.42. Show how it is forwarded.
6. A packet arrives at router R1 in Figure 6.8 with destination address 145.80.14.26. Show how it is forwarded.
7. A packet arrives at router R1 in Figure 6.8 with destination address 147.26.50.30. Show how it is forwarded.
8. A packet arrives at the router in Figure 6.11 with destination address 145.14.192.71. Show how it is forwarded.
9. A packet arrives at the router in Figure 6.11 with destination address 135.11.80.21. Show how it is forwarded.
10. A packet arrives at router R1 in Figure 6.13 with destination address 201.4.16.70. Show how it is forwarded.
11. A packet arrives at router R1 in Figure 6.13 with destination address 202.70.20.30. Show how it is forwarded.
12. Show a routing table for a host that is totally isolated.
13. Show a routing table for a host that is connected to a LAN without being connected to the Internet.
14. Find the topology of the network if Table 6.3 is the routing table for router R1.

**Table 6.3** *Routing table for Exercise 14*

| *Mask* | *Network Address* | *Next-Hop Address* | *Interface* |
|---|---|---|---|
| /27 | 202.14.17.224 | ---- | m1 |
| /18 | 145.23.192.0 | ---- | m0 |
| default | default | 130.56.12.4 | m2 |

15. Can router R1 in Figure 6.16 receive a packet with destination address 140.24.7.194? Explain your answer.
16. Can router R1 in Figure 6.16 receive a packet with destination address 140.24.7.42? Explain your answer.
17. Show the routing table for regional ISP in Figure 6.17.
18. Show the routing table for local ISP 1 in Figure 6.17.
19. Show the routing table for local ISP 2 in Figure 6.17.
20. Show the routing table for local ISP 3 in Figure 6.17.
21. Show the routing table for small ISP 1 in Figure 6.17.

## Research Activities

22. Show how an MPLS packet is encapsulated in a frame. Find out what should be the value of the type field when the protocol is the Ethernet.
23. Compare Multi-Layer Switching (MLS) used by Cisco Systems Inc. with MPLS technology we described here.
24. Some people argue that the MPLS should be called layer 2.5 in the TCP/IP protocol suite. Do you agree? Explain.
25. Find how your ISP uses address aggregation and longest mask match principles.
26. Find whether or not your IP address is part of the geographical address allocation.
27. If you are using a router, find the number and names of the columns in the routing table.
28. Cisco is one of the dominant manufacturers of routers. Find information about the different types of routers manufactured by this company.

CHAPTER

# 7

# *Internet Protocol Version 4 (IPv4)*

After discussing the IP addressing mechanism and delivery and forwarding of IP packets, we discuss the format of the IP packet in this chapter. We show how an IP packet is made of a base header and options that sometimes are used to facilitate or control the delivery of packets.

## OBJECTIVES

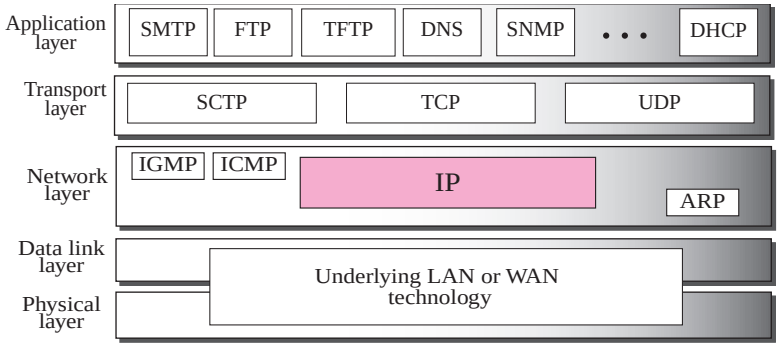
*The chapter has several objectives:*

- To explain the general idea behind the IP protocol and show the position of IP in relation to other protocols in TCP/IP protocol suite.
- To show the general format of an IPv4 datagram and list the fields in the header.
- To discuss fragmentation and reassembly of datagrams and how the original datagram can be recovered out of fragmented ones.
- To discuss several options that can be in an IPv4 datagram and their applications.
- To discuss some reserved blocks in the address space and their applications.
- To show how a checksum is calculated for the header of an IPv4 datagram at the sending site and how the checksum is checked at the receiver site.
- To discuss IP over ATM and compare it with IP over LANs and/or point-to-point WANs.
- To show a simplified version of the IP package and give the pseudocode for some modules.

## 7.1 INTRODUCTION

The **Internet Protocol (IP)** is the transmission mechanism used by the TCP/IP protocols at the network layer. Figure 7.1 shows the position of IP in the suite.

**Figure 7.1** *Position of IP in TCP/IP protocol suite*




IP is an unreliable and connectionless datagram protocol—a **best-effort delivery** service. The term *best-effort* means that IP packets can be corrupted, lost, arrive out of order, or delayed and may create congestion for the network.

If reliability is important, IP must be paired with a reliable protocol such as TCP. An example of a more commonly understood best-effort delivery service is the post office. The post office does its best to deliver the mail but does not always succeed. If an unregistered letter is lost, it is up to the sender or would-be recipient to discover the loss and rectify the problem. The post office itself does not keep track of every letter and cannot notify a sender of loss or damage.

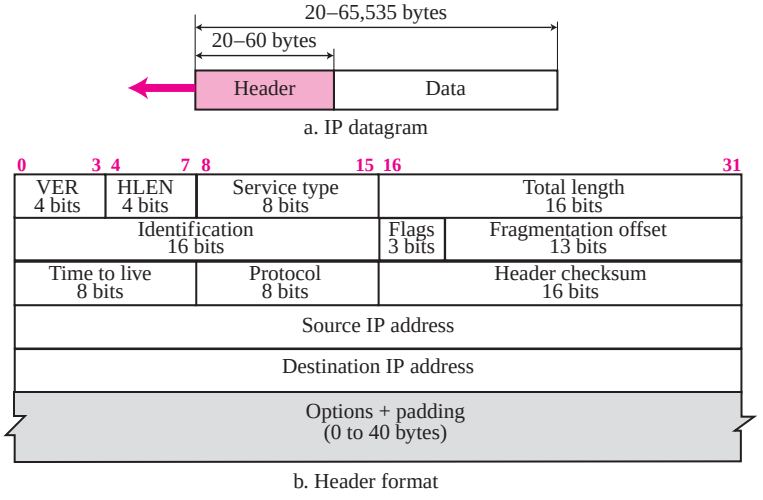
IP is also a connectionless protocol for a packet switching network that uses the datagram approach (see Chapter 4). This means that each datagram is handled independently, and each datagram can follow a different route to the destination. This implies that datagrams sent by the same source to the same destination could arrive out of order. Also, some could be lost or corrupted during transmission. Again, IP relies on a higher-level protocol to take care of all these problems.

## 7.2 DATAGRAMS

Packets in the network (internet) layer are called **datagrams.** Figure 7.2 shows the IP datagram format. A datagram is a variable-length packet consisting of two parts: header and data. The header is 20 to 60 bytes in length and contains information essential to

routing and delivery. It is customary in TCP/IP to show the header in 4-byte sections. A brief description of each field is in order.

**Figure 7.2** *IP datagram*

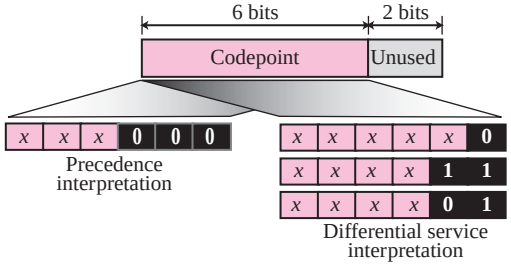



- **Version (VER).** This 4-bit field defines the version of the IP protocol. Currently the version is 4. However, version 6 (or IPv6) may totally replace version 4 in the future. This field tells the IP software running in the processing machine that the datagram has the format of version 4. All fields must be interpreted as specified in the fourth version of the protocol. If the machine is using some other version of IP, the datagram is discarded rather than interpreted incorrectly.
- **Header length (HLEN).** This 4-bit field defines the total length of the datagram header in 4-byte words. This field is needed because the length of the header is variable (between 20 and 60 bytes). When there are no options, the **header length** is 20 bytes, and the value of this field is 5 ($5 \times 4 = 20$). When the option field is at its maximum size, the value of this field is 15 ($15 \times 4 = 60$).
- **Service type.** In the original design of IP header, this field was referred to as **type of service (TOS)**, which defined how the datagram should be handled. Part of the field was used to define the precedence of the datagram; the rest defined the type of service (low delay, high throughput, and so on). IETF has changed the interpretation of this 8-bit field. This field now defines a set of **differentiated services.** The new interpretation is shown in Figure 7.3.

  In this interpretation, the first 6 bits make up the **codepoint** subfield and the last 2 bits are not used. The codepoint subfield can be used in two different ways.

  **a.** When the 3 right-most bits are 0s, the 3 left-most bits are interpreted the same as the precedence bits in the service type interpretation. In other words, it is compatible with the old interpretation. The precedence defines the eight-level

**Figure 7.3** *Service type*




priority of the datagram (0 to 7) in issues such as congestion. If a router is congested and needs to discard some datagrams, those datagrams with lowest precedence are discarded first. Some datagrams in the Internet are more important than the others. For example, a datagram used for network management is much more urgent and important than a datagram containing optional information for a group.

**b.** When the 3 right-most bits are not all 0s, the 6 bits define 56 (64 – 8) services based on the priority assignment by the Internet or local authorities according to Table 7.1. The first category contains 24 service types; the second and the third each contain 16. The first category is assigned by the Internet authorities (IETF). The second category can be used by local authorities (organizations). The third category is temporary and can be used for experimental purposes. Note that these assignments have not yet been finalized.

**Table 7.1** *Values for codepoints*

| *Category* | *Codepoint* | *Assigning Authority* |
|---|---|---|
| 1 | XXXXX0 | Internet |
| 2 | XXXX11 | Local |
| 3 | XXXX01 | Temporary or experimental |

❑ **Total length.** This is a 16-bit field that defines the total length (header plus data) of the IP datagram in bytes. To find the length of the data coming from the upper layer, subtract the header length from the total length. The header length can be found by multiplying the value in the HLEN field by four.

Length of data = total length – header length

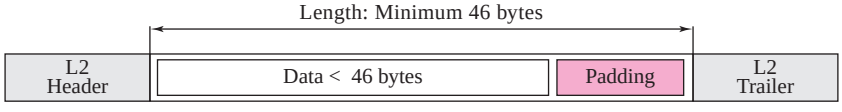
Since the field length is 16 bits, the total length of the IP datagram is limited to 65,535 ($2^{16} - 1$) bytes, of which 20 to 60 bytes are the header and the rest is data from the upper layer.

**The *total length* field defines the total length of the datagram including the header.**

Though a size of 65,535 bytes might seem large, the size of the IP datagram may increase in the near future as the underlying technologies allow even more throughput (more bandwidth). When we discuss fragmentation in the next section, we will see that some physical networks are not able to encapsulate a datagram of 65,535 bytes in their frames. The datagram must be fragmented to be able to pass through those networks.

One may ask why we need this field anyway. When a machine (router or host) receives a frame, it drops the header and the trailer leaving the datagram. Why include an extra field that is not needed? The answer is that in many cases we really do not need the value in this field. However, there are occasions in which the datagram is not the only thing encapsulated in a frame; it may be that padding has been added. For example, the Ethernet protocol has a minimum and maximum restriction on the size of data that can be encapsulated in a frame (46 to 1500 bytes). If the size of an IP datagram is less than 46 bytes, some padding will be added to meet this requirement. In this case, when a machine decapsulates the datagram, it needs to check the total length field to determine how much is really data and how much is padding (see Figure 7.4).

**Figure 7.4** *Encapsulation of a small datagram in an Ethernet frame*




- **Identification.** This field is used in fragmentation (discussed in the next section).
- **Flags.** This field is used in fragmentation (discussed in the next section).
- **Fragmentation offset.** This field is used in fragmentation (discussed in the next section).
- **Time to live.** A datagram has a limited lifetime in its travel through an internet. This field was originally designed to hold a timestamp, which was decremented by each visited router. The datagram was discarded when the value became zero. However, for this scheme, all the machines must have synchronized clocks and must know how long it takes for a datagram to go from one machine to another. Today, this field is mostly used to control the maximum number of hops (routers) visited by the datagram. When a source host sends the datagram, it stores a number in this field. This value is approximately two times the maximum number of routes between any two hosts. Each router that processes the datagram decrements this number by one. If this value, after being decremented, is zero, the router discards the datagram.

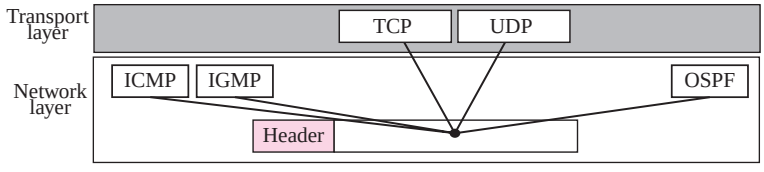
  This field is needed because routing tables in the Internet can become corrupted. A datagram may travel between two or more routers for a long time without ever getting delivered to the destination host. This field limits the lifetime of a datagram.

  Another use of this field is to intentionally limit the journey of the packet. For example, if the source wants to confine the packet to the local network, it can store

1 in this field. When the packet arrives at the first router, this value is decremented to 0, and the datagram is discarded.

- **Protocol.** This 8-bit field defines the higher-level protocol that uses the services of the IP layer. An IP datagram can encapsulate data from several higher level protocols such as TCP, UDP, ICMP, and IGMP. This field specifies the final destination protocol to which the IP datagram should be delivered. In other words, since the IP protocol multiplexes and demultiplexes data from different higher-level protocols, the value of this field helps in the demultiplexing process when the datagram arrives at its final destination (see Figure 7.5).

**Figure 7.5** *Multiplexing*




Some of the value of this field for different higher-level protocols is shown in Table 7.2.

**Table 7.2** *Protocols*

| *Value* | *Protocol* | *Value* | *Protocol* |
|---|---|---|---|
| 1 | ICMP | 17 | UDP |
| 2 | IGMP | 89 | OSPF |
| 6 | TCP | | |

- **Checksum.** The checksum concept and its calculation are discussed later in this chapter.
- **Source address.** This 32-bit field defines the IP address of the source. This field must remain unchanged during the time the IP datagram travels from the source host to the destination host.
- **Destination address.** This 32-bit field defines the IP address of the destination. This field must remain unchanged during the time the IP datagram travels from the source host to the destination host.

### Example 7.1

An IP packet has arrived with the first 8 bits as shown:

```
01000010
```

The receiver discards the packet. Why?

#### Solution

There is an error in this packet. The 4 left-most bits (0100) show the version, which is correct. The next 4 bits (0010) show the wrong header length ($2 \times 4 = 8$). The minimum number of bytes in the header must be 20. The packet has been corrupted in transmission.

### Example 7.2

In an IP packet, the value of HLEN is 1000 in binary. How many bytes of options are being carried by this packet?

**Solution**

The HLEN value is 8, which means the total number of bytes in the header is $8 \times 4$ or 32 bytes. The first 20 bytes are the base header, the next 12 bytes are the options.

### Example 7.3

In an IP packet, the value of HLEN is $5_{16}$ and the value of the total length field is $0028_{16}$. How many bytes of data are being carried by this packet?

**Solution**

The HLEN value is 5, which means the total number of bytes in the header is $5 \times 4$ or 20 bytes (no options). The total length is 40 bytes, which means the packet is carrying 20 bytes of data ($40 - 20$).

### Example 7.4

An IP packet has arrived with the first few hexadecimal digits as shown below:

```
45000028000100000102 . . .
```

How many hops can this packet travel before being dropped? The data belong to what upper layer protocol?

**Solution**

To find the time-to-live field, we skip 8 bytes (16 hexadecimal digits). The time-to-live field is the ninth byte, which is 01. This means the packet can travel only one hop. The protocol field is the next byte (02), which means that the upper layer protocol is IGMP (see Table 7.2).

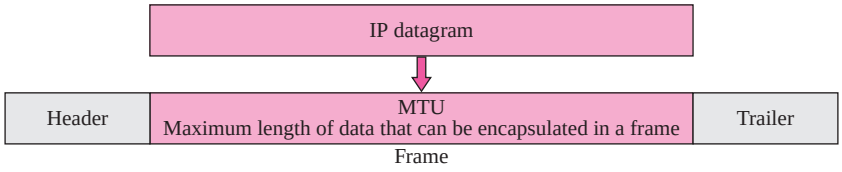
## 7.3 FRAGMENTATION

A datagram can travel through different networks. Each router decapsulates the IP datagram from the frame it receives, processes it, and then encapsulates it in another frame. The format and size of the received frame depend on the protocol used by the physical network through which the frame has just traveled. The format and size of the sent frame depend on the protocol used by the physical network through which the frame is going to travel. For example, if a router connects a LAN to a WAN, it receives a frame in the LAN format and sends a frame in the WAN format.

### Maximum Transfer Unit (MTU)

Each data link layer protocol has its own frame format in most protocols. One of the fields defined in the format is the maximum size of the data field. In other words, when a datagram is encapsulated in a frame, the total size of the datagram must be less than this maximum size, which is defined by the restrictions imposed by the hardware and software used in the network (see Figure 7.6).

**Figure 7.6** *MTU*




The value of the MTU differs from one physical network protocol to another. For example, the value for the Ethernet LAN is 1500 bytes, for FDDI LAN is 4352 bytes, and for PPP is 296 bytes.

In order to make the IP protocol independent of the physical network, the designers decided to make the maximum length of the IP datagram equal to 65,535 bytes. This makes transmission more efficient if we use a protocol with an MTU of this size. However, for other physical networks, we must divide the datagram to make it possible to pass through these networks. This is called **fragmentation.**

The source usually does not fragment the IP packet. The transport layer will instead segment the data into a size that can be accommodated by IP and the data link layer in use.

When a datagram is fragmented, each fragment has its own header with most of the fields repeated, but some changed. A fragmented datagram may itself be fragmented if it encounters a network with an even smaller MTU. In other words, a datagram can be fragmented several times before it reaches the final destination.

A datagram can be fragmented by the source host or any router in the path. The reassembly of the datagram, however, is done only by the destination host because each fragment becomes an independent datagram. Whereas the fragmented datagram can travel through different routes, and we can never control or guarantee which route a fragmented datagram may take, all of the fragments belonging to the same datagram should finally arrive at the destination host. So it is logical to do the reassembly at the final destination. An even stronger objection for reassembling packets during the transmission is the loss of efficiency it incurs.

When a datagram is fragmented, required parts of the header must be copied by all fragments. The option field may or may not be copied as we will see in the next section. The host or router that fragments a datagram must change the values of three fields: flags, fragmentation offset, and total length. The rest of the fields must be copied. Of course, the value of the checksum must be recalculated regardless of fragmentation.

**Only data in a datagram is fragmented.**

## Fields Related to Fragmentation

The fields that are related to fragmentation and reassembly of an IP datagram are the identification, flags, and fragmentation offset fields.

- **Identification.** This 16-bit field identifies a datagram originating from the source host. The combination of the identification and source IP address must uniquely

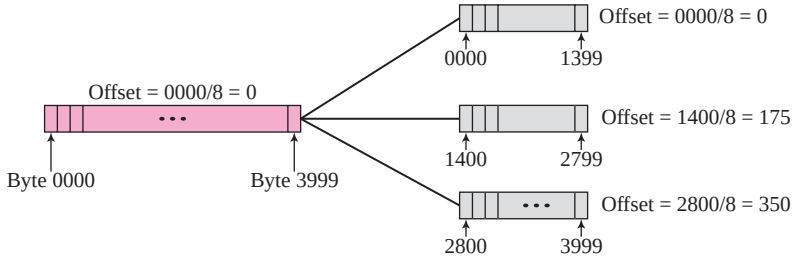
define a datagram as it leaves the source host. To guarantee uniqueness, the IP protocol uses a counter to label the datagrams. The counter is initialized to a positive number. When the IP protocol sends a datagram, it copies the current value of the counter to the identification field and increments the counter by one. As long as the counter is kept in the main memory, uniqueness is guaranteed. When a datagram is fragmented, the value in the identification field is copied into all fragments. In other words, all fragments have the same identification number, which is also the same as the original datagram. The identification number helps the destination in reassembling the datagram. It knows that all fragments having the same identification value should be assembled into one datagram.

- **Flags.** This is a three-bit field. The first bit is reserved (not used). The second bit is called the *do not fragment* bit. If its value is 1, the machine must not fragment the datagram. If it cannot pass the datagram through any available physical network, it discards the datagram and sends an ICMP error message to the source host (see Chapter 9). If its value is 0, the datagram can be fragmented if necessary. The third bit is called the *more fragment* bit. If its value is 1, it means the datagram is not the last fragment; there are more fragments after this one. If its value is 0, it means this is the last or only fragment (see Figure 7.7).

**Figure 7.7** *Flags field*


- **Fragmentation offset.** This 13-bit field shows the relative position of this fragment with respect to the whole datagram. It is the offset of the data in the original datagram measured in units of 8 bytes. Figure 7.8 shows a datagram with a data size of 4000 bytes fragmented into three fragments. The bytes in the original datagram are numbered 0 to 3999. The first fragment carries bytes 0 to 1399. The offset for this datagram is 0/8 = 0. The second fragment carries bytes 1400 to 2799; the offset value for this fragment is 1400/8 = 175. Finally, the third fragment carries bytes 2800 to 3999. The offset value for this fragment is 2800/8 = 350.

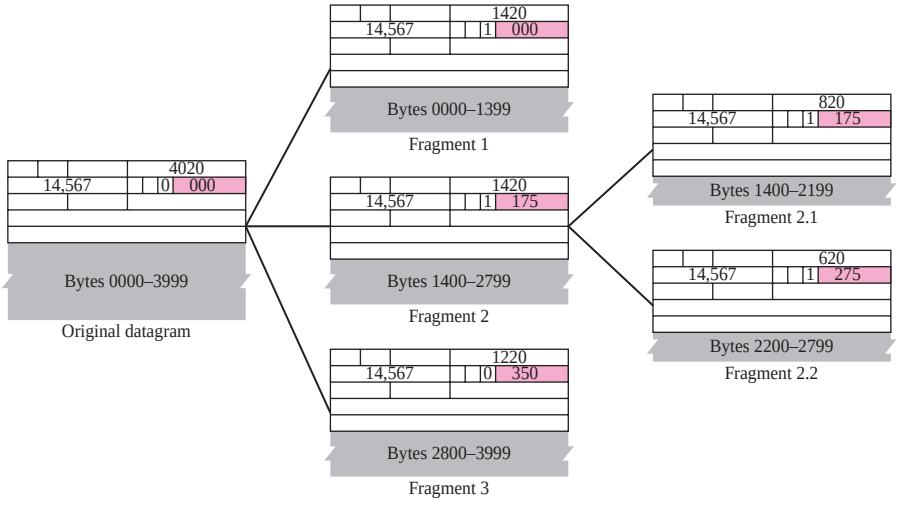
**Figure 7.8** *Fragmentation example*



Remember that the value of the offset is measured in units of 8 bytes. This is done because the length of the offset field is only 13 bits long and cannot represent a sequence of bytes greater than 8191. This forces hosts or routers that fragment datagrams to choose the size of each fragment so that the first byte number is divisible by 8.

Figure 7.9 shows an expanded view of the fragments in the previous figure. Notice the value of the identification field is the same in all fragments. Notice the value of the flags field with the *more* bit set for all fragments except the last. Also, the value of the offset field for each fragment is shown.

**Figure 7.9** *Detailed fragmentation example*




The figure also shows what happens if a fragment itself is fragmented. In this case the value of the offset field is always relative to the original datagram. For example, in the figure, the second fragment is itself fragmented later to two fragments of 800 bytes and 600 bytes, but the offset shows the relative position of the fragments to the original data.

It is obvious that even if each fragment follows a different path and arrives out of order, the final destination host can reassemble the original datagram from the fragments received (if none of them is lost) using the following strategy:

**a.** The first fragment has an offset field value of zero.

**b.** Divide the length of the first fragment by 8. The second fragment has an offset value equal to that result.

**c.** Divide the total length of the first and second fragment by 8. The third fragment has an offset value equal to that result.

**d.** Continue the process. The last fragment has a *more* bit value of 0.

### Example 7.5

A packet has arrived with an *M* bit value of 0. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

**Solution**

If the *M* bit is 0, it means that there are no more fragments; the fragment is the last one. However, we cannot say if the original packet was fragmented or not. A nonfragmented packet is considered the last fragment.

### Example 7.6

A packet has arrived with an *M* bit value of 1. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

**Solution**

If the *M* bit is 1, it means that there is at least one more fragment. This fragment can be the first one or a middle one, but not the last one. We don't know if it is the first one or a middle one; we need more information (the value of the fragmentation offset). See also the next example.

### Example 7.7

A packet has arrived with an *M* bit value of 1 and a fragmentation offset value of zero. Is this the first fragment, the last fragment, or a middle fragment?

**Solution**

Because the *M* bit is 1, it is either the first fragment or a middle one. Because the offset value is 0, it is the first fragment.

### Example 7.8

A packet has arrived in which the offset value is 100. What is the number of the first byte? Do we know the number of the last byte?

**Solution**

To find the number of the first byte, we multiply the offset value by 8. This means that the first byte number is 800. We cannot determine the number of the last byte unless we know the length of the data.

### Example 7.9

A packet has arrived in which the offset value is 100, the value of HLEN is 5 and the value of the total length field is 100. What is the number of the first byte and the last byte?

**Solution**

The first byte number is $100 \times 8 = 800$. The total length is 100 bytes and the header length is 20 bytes ($5 \times 4$), which means that there are 80 bytes in this datagram. If the first byte number is 800, the last byte number must be 879.

## 7.4 OPTIONS

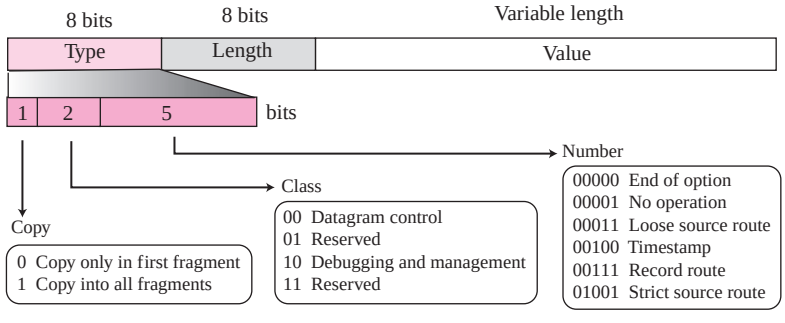
The header of the IP datagram is made of two parts: a fixed part and a variable part. The fixed part is 20 bytes long and was discussed in the previous section. The variable part comprises the options, which can be a maximum of 40 bytes.

Options, as the name implies, are not required for a datagram. They can be used for network testing and debugging. Although options are not a required part of the IP header, option processing is required of the IP software. This means that all implementations must be able to handle options if they are present in the header.

### Format

Figure 7.10 shows the format of an option. It is composed of a 1-byte type field, a 1-byte length field, and a variable-sized value field. The three fields are often referred to as type-length-value or TLV.

**Figure 7.10** *Option format*




#### *Type*

The **type field** is 8 bits long and contains three subfields: copy, class, and number.

- **Copy.** This 1-bit subfield controls the presence of the option in fragmentation. When its value is 0, it means that the option must be copied only to the first fragment. If its value is 1, it means the option must be copied to all fragments.
- **Class.** This 2-bit subfield defines the general purpose of the option. When its value is 00, it means that the option is used for datagram control. When its value is 10, it means that the option is used for debugging and management. The other two possible values (01 and 11) have not yet been defined.
- **Number.** This 5-bit subfield defines the type of option. Although 5 bits can define up to 32 different types, currently only 6 types are in use. These will be discussed in a later section.

### *Length*

The **length field** defines the total length of the option including the type field and the length field itself. This field is not present in all of the option types.

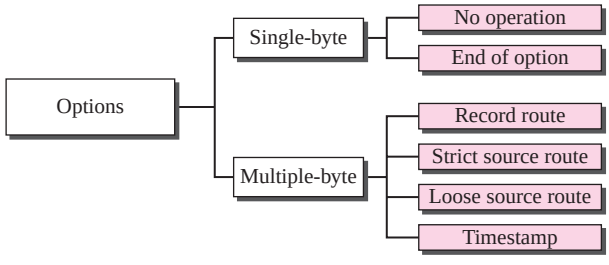
### *Value*

The **value field** contains the data that specific options require. Like the length field, this field is also not present in all option types.

## Option Types

As mentioned previously, only six options are currently being used. Two of these are 1-byte options, and they do not require the length or the data fields. Four of them are multiple-byte options; they require the length and the data fields (see Figure 7.11).

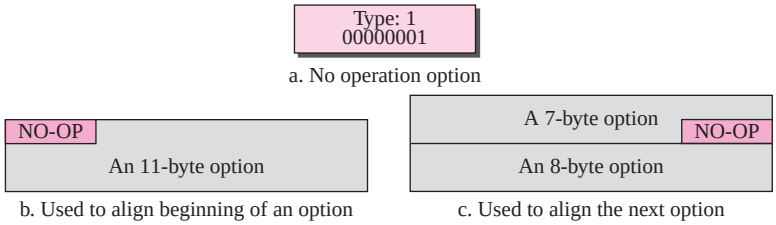
**Figure 7.11** *Categories of options*




### *No-Operation Option*

A **no-operation option** is a 1-byte option used as a filler between options. For example, it can be used to align the next option on a 16-bit or 32-bit boundary (see Figure 7.12).

**Figure 7.12** *No operation option*

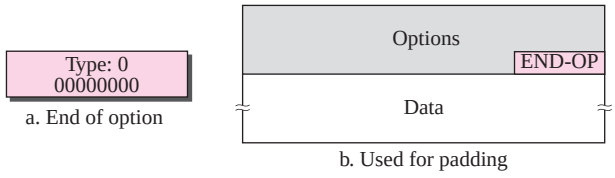



### *End-of-Option Option*

An **end-of-option option** is also a 1-byte option used for padding at the end of the option field. It, however, can only be used as the last option. Only one end-of-option option can be used. After this option, the receiver looks for the payload data. This

means that if more than 1 byte is needed to align the option field, some no-operation options must be used, followed by an end-of-option option (see Figure 7.13).

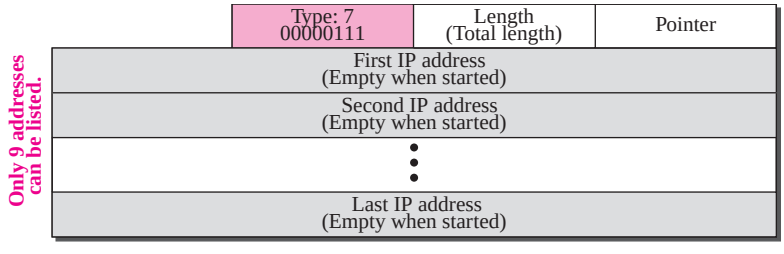
**Figure 7.13** *End-of-option option*




### *Record-Route Option*

A **record-route option** is used to record the Internet routers that handle the datagram. It can list up to nine router IP addresses since the maximum size of the header is 60 bytes, which must include 20 bytes for the base header. This implies that only 40 bytes are left over for the option part. The source creates placeholder fields in the option to be filled by the visited routers. Figure 7.14 shows the format of the record route option.

**Figure 7.14** *Record-route option*

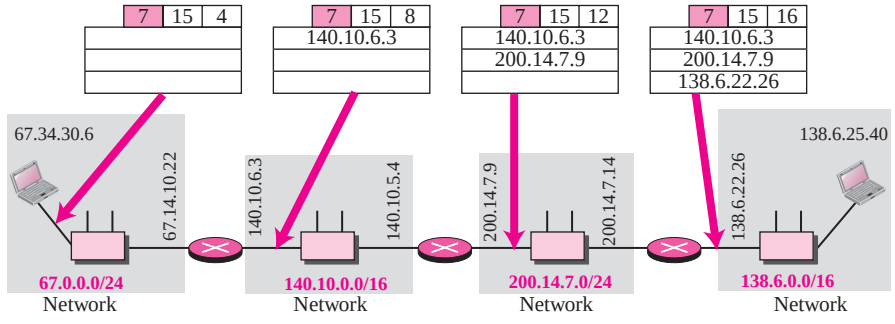



Both the code and length fields have been described above. The **pointer field** is an offset integer field containing the byte number of the first empty entry. In other words, it points to the first available entry.

The source creates empty fields for the IP addresses in the data field of the option. When the datagram leaves the source, all of the fields are empty. The pointer field has a value of 4, pointing to the first empty field.

When the datagram is traveling, each router that processes the datagram compares the value of the pointer with the value of the length. If the value of the pointer is greater than the value of the length, the option is full and no changes are made. However, if the value of the pointer is not greater than the value of the length, the router inserts its outgoing IP address in the next empty field (remember that a router has more than one IP address). In this case, the router adds the IP address of its interface from which the datagram is leaving. The router then increments the value of the pointer by 4. Figure 7.15 shows the entries as the datagram travels left to right from router to router.

**Figure 7.15** *Record-route concept*



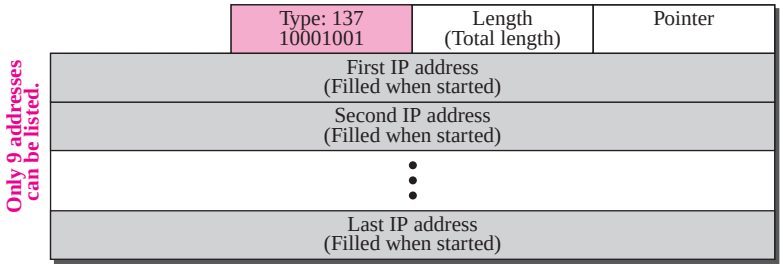

### *Strict-Source-Route Option*

A **strict-source-route option** is used by the source to predetermine a route for the datagram as it travels through the Internet. Dictation of a route by the source can be useful for several purposes. The sender can choose a route with a specific type of service, such as minimum delay or maximum throughput. Alternatively, it may choose a route that is safer or more reliable for the sender's purpose. For example, a sender can choose a route so that its datagram does not travel through a competitor's network.

If a datagram specifies a strict source route, all of the routers defined in the option must be visited by the datagram. A router must not be visited if its IP address is not listed in the datagram. If the datagram visits a router that is not on the list, the datagram is discarded and an error message is issued. If the datagram arrives at the destination and some of the entries were not visited, it will also be discarded and an error message issued.

Regular users of the Internet, however, are not usually aware of the physical topology of the Internet. Consequently, strict source routing is not the choice of most users. Figure 7.16 shows the format of the strict source route option.

**Figure 7.16** *Strict-source-route option*

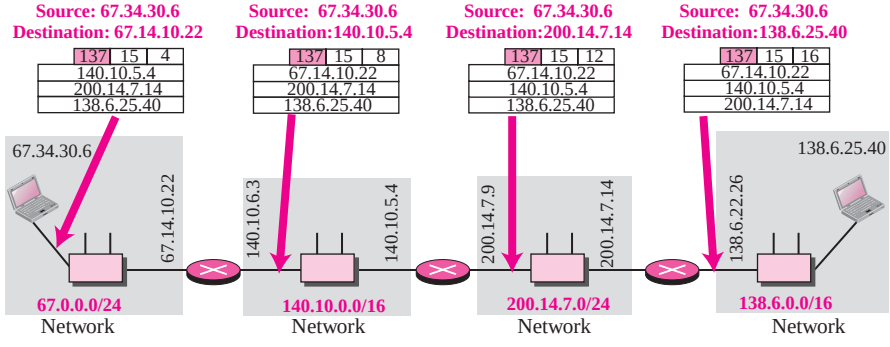



The format is similar to the record route option with the exception that all of the IP addresses are entered by the sender.

When the datagram is traveling, each router that processes the datagram compares the value of the pointer with the value of the length. If the value of the pointer is greater

than the value of the length, the datagram has visited all of the predefined routers. The datagram cannot travel anymore; it is discarded and an error message is created. If the value of the pointer is not greater than the value of the length, the router compares the destination IP address with its incoming IP address: If they are equal, it processes the datagram, swaps the IP address pointed by the pointer with the destination address, increments the pointer value by 4, and forwards the datagram. If they are not equal, it discards the datagram and issues an error message. Figure 7.17 shows the actions taken by each router as a datagram travels from source to destination.

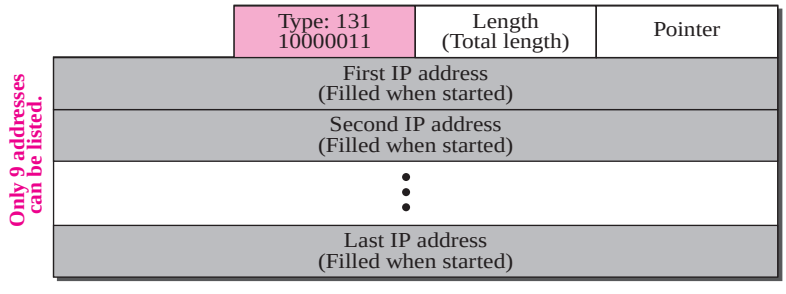
**Figure 7.17** *Strict-source-route concept*




### *Loose-Source-Route Option*

A **loose-source-route option** is similar to the strict source route, but it is more relaxed. Each router in the list must be visited, but the datagram can visit other routers as well. Figure 7.18 shows the format of the loose source route option.

**Figure 7.18** *Loose-source-route option*



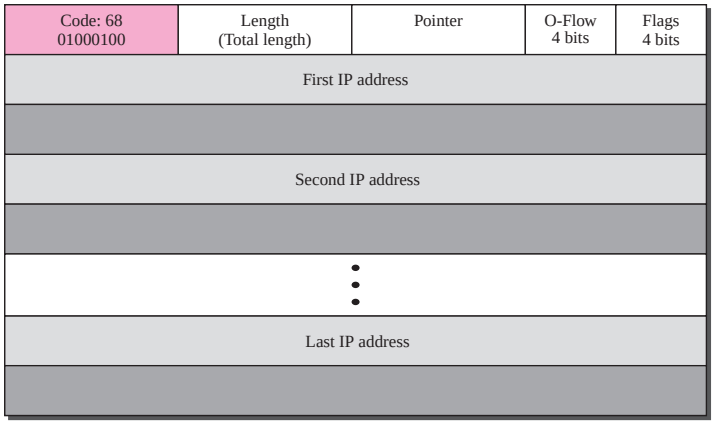

### *Timestamp*

A **timestamp option** is used to record the time of datagram processing by a router. The time is expressed in milliseconds from midnight, Universal Time. Knowing the time a

datagram is processed can help users and managers track the behavior of the routers in the Internet. We can estimate the time it takes for a datagram to go from one router to another. We say *estimate* because, although all routers may use Universal Time, their local clocks may not be synchronized.

However, nonprivileged users of the Internet are not usually aware of the physical topology of the Internet. Consequently, a timestamp option is not a choice for most users. Figure 7.19 shows the format of the timestamp option.

**Figure 7.19** *Timestamp option*




In this figure, the definitions of the code and length fields are the same as before. The overflow field records the number of routers that could not add their timestamp because no more fields were available. The flags field specifies the visited router responsibilities. If the flag value is 0, each router adds only the timestamp in the provided field. If the flag value is 1, each router must add its outgoing IP address and the timestamp. If the value is 3, the IP addresses are given, and each router must check the given IP address with its own incoming IP address. If there is a match, the router overwrites the IP address with its outgoing IP address and adds the timestamp (see Figure 7.20).

**Figure 7.20** *Use of flag in timestamp*

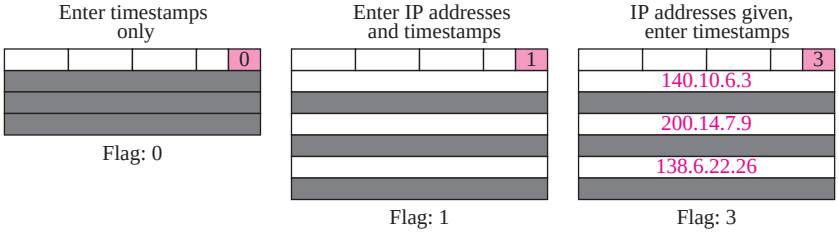
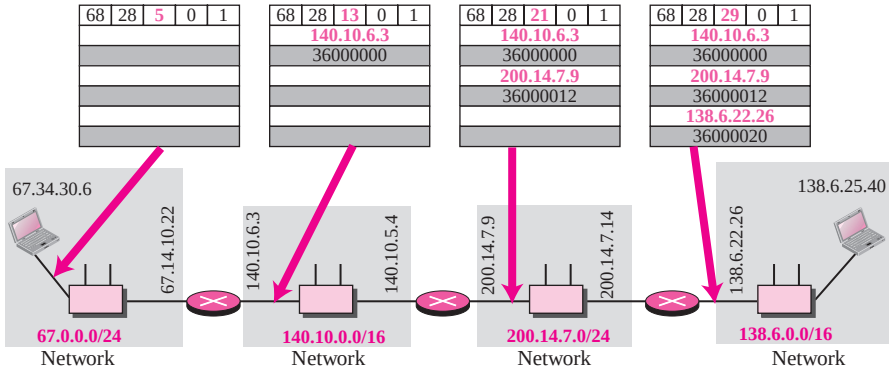


Figure 7.21 shows the actions taken by each router when a datagram travels from source to destination. The figure assumes a flag value of 1.

**Figure 7.21** *Timestamp concept*




## Example 7.10

Which of the six options must be copied to each fragment?

### Solution

We look at the first (left-most) bit of the type for each option.

- **a.** No operation: type is **0**0000001; not copied.
- **b.** End of option: type is **0**0000000; not copied.
- **c.** Record route: type is **0**0000111; not copied.
- **d.** Strict source route: type is **1**0001001; copied.
- **e.** Loose source route: type is **1**0000011; copied.
- **f.** Timestamp: type is **0**1000100; not copied.

## Example 7.11

Which of the six options are used for datagram control and which are used for debugging and management?

### Solution

We look at the second and third (left-most) bits of the type.

- **a.** No operation: type is 0**00**00001; datagram control.
- **b.** End of option: type is 0**00**00000; datagram control.
- **c.** Record route: type is 0**00**00111; datagram control.
- **d.** Strict source route: type is 1**00**01001; datagram control.
- **e.** Loose source route: type is 1**00**00011; datagram control.
- **f.** Timestamp: type is 0**10**00100; debugging and management control.

### Example 7.12

One of the utilities available in UNIX to check the traveling of the IP packets is **ping.** In the next chapter, we talk about the *ping* program in more detail. In this example, we want to show how to use the program to see if a host is available. We ping a server at De Anza College named *fhda.edu*. The result shows that the IP address of the host is 153.18.8.1. The result also shows the number of bytes used.

```
$  ping fhda.edu
PING fhda.edu (153.18.8.1) 56(84) bytes of data.
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq =
0 ttl=62 time=1.87 ms
...
```

### Example 7.13

We can also use the *ping* utility with the -R option to implement the record route option. The result shows the interfaces and IP addresses.

```
$ ping -R fhda.edu
PING fhda.edu (153.18.8.1) 56(124) bytes of data.
64 bytes from tiptoe.fhda.edu
(153.18.8.1): icmp_seq=0 ttl=62 time=2.70 ms
RR:  voyager.deanza.fhda.edu (153.18.17.11)
     Dcore_G0_3-69.fhda.edu (153.18.251.3)
     Dbackup_V13.fhda.edu (153.18.191.249)
     tiptoe.fhda.edu (153.18.8.1)
     Dbackup_V62.fhda.edu (153.18.251.34)
     Dcore_G0_1-6.fhda.edu (153.18.31.254)
     voyager.deanza.fhda.edu (153.18.17.11)
```

### Example 7.14

The **traceroute** utility can also be used to keep track of the route of a packet. The result shows the three routers visited.

```
$ traceroute fhda.edu
traceroute to fhda.edu (153.18.8.1), 30 hops max, 38 byte packets
 1 Dcore_G0_1-6.fhda.edu (153.18.31.254)  0.972 ms  0.902 ms
   0.881 ms
 2 Dbackup_V69.fhda.edu (153.18.251.4)  2.113 ms  1.996 ms
   2.059 ms
 3 tiptoe.fhda.edu (153.18.8.1)  1.791 ms  1.741 ms  1.751 ms
```

### Example 7.15

The traceroute program can be used to implement loose source routing. The -g option allows us to define the routers to be visited, from the source to destination. The following shows how we can send a packet to the fhda.edu server with the requirement that the packet visit the router 153.18.251.4.

```
$ traceroute -g  153.18.251.4 fhda.edu.
traceroute to fhda.edu (153.18.8.1), 30 hops max, 46 byte packets
 1  Dcore_G0_1-6.fhda.edu (153.18.31.254)  0.976 ms  0.906 ms
    0.889 ms
 2  Dbackup_V69.fhda.edu (153.18.251.4)  2.168 ms  2.148 ms
    2.037 ms
```

### Example 7.16

The traceroute program can also be used to implement strict source routing. The -G option forces the packet to visit the routers defined in the command line. The following shows how we can send a packet to the *fhda.edu* server and force the packet to visit only the router 153.18.251.4.

```
$ traceroute -G  153.18.251.4 fhda.edu.
traceroute to fhda.edu (153.18.8.1), 30 hops max, 46 byte packets
  1  Dbackup_V69.fhda.edu (153.18.251.4)  2.168 ms  2.148 ms
     2.037 ms
```

## 7.5 CHECKSUM

The error detection method used by most TCP/IP protocols is called the **checksum.** The checksum protects against the corruption that may occur during the transmission of a packet. It is redundant information added to the packet.

The checksum is calculated at the sender and the value obtained is sent with the packet. The receiver repeats the same calculation on the whole packet including the checksum. If the result is satisfactory (see below), the packet is accepted; otherwise, it is rejected.

### Checksum Calculation at the Sender

At the sender, the packet header is divided into *n*-bit sections (*n* is usually 16). These sections are added together using one's complement arithmetic (see Appendix D), resulting in a sum that is also *n* bits long. The sum is then complemented (all 0s changed to 1s and all 1s to 0s) to produce the checksum.

**To create the checksum the sender does the following:**

- The packet is divided into *k* sections, each of *n* bits.
- All sections are added together using one's complement arithmetic.
- The final result is complemented to make the checksum.

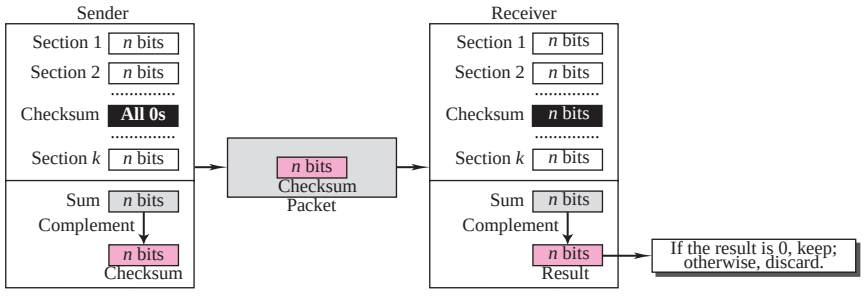
### Checksum Calculation at the Receiver

The receiver divides the received packet into *k* sections and adds all sections. It then complements the result. If the final result is 0, the packet is accepted; otherwise, it is rejected. Figure 7.22 shows graphically what happens at the sender and the receiver.

We said when the receiver adds all of the sections and complements the result, it should get zero if there is no error in the data during transmission or processing. This is true because of the rules in one's complement arithmetic.

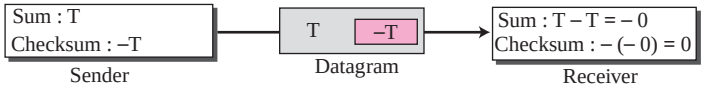
Assume that we get a number called *T* when we add all the sections in the sender. When we complement the number in one's complement arithmetic, we get the negative of the number. This means that if the sum of all sections is *T*, the checksum is $-T$.

**Figure 7.22** *Checksum concept*




When the receiver receives the packet, it adds all the sections. It adds $T$ and $-T$ which, in one's complement, is $-0$ (minus zero). When the result is complemented, $-0$ becomes 0. Thus if the final result is 0, the packet is accepted; otherwise, it is rejected (see Figure 7.23).

**Figure 7.23** *Checksum in one's complement arithmetic*




## Checksum in the IP Packet

The implementation of the checksum in the IP packet follows the same principles discussed above. First, the value of the checksum field is set to 0. Then, the entire header is divided into 16-bit sections and added together. The result (sum) is complemented and inserted into the checksum field.

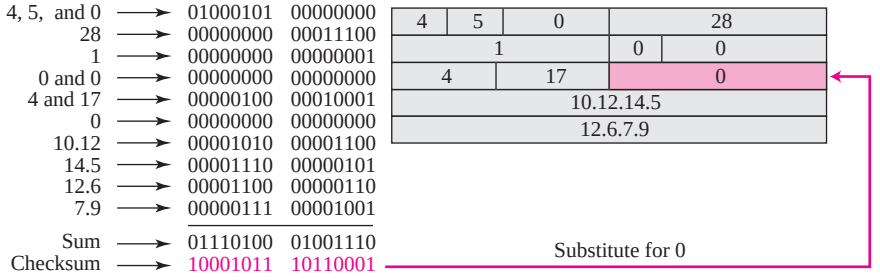
The checksum in the IP packet covers only the header, not the data. There are two good reasons for this. First, all higher-level protocols that encapsulate data in the IP datagram have a checksum field that covers the whole packet. Therefore, the checksum for the IP datagram does not have to check the encapsulated data. Second, the header of the IP packet changes with each visited router, but the data do not. So the checksum includes only the part that has changed. If the data were included, each router would have to recalculate the checksum for the whole packet, which means an increase in processing time.

**Checksum in IP covers only the header, not the data.**

### Example 7.17

Figure 7.24 shows an example of a checksum calculation at the sender site for an IP header without options. The header is divided into 16-bit sections. All the sections are added and the sum is complemented. The result is inserted in the checksum field.

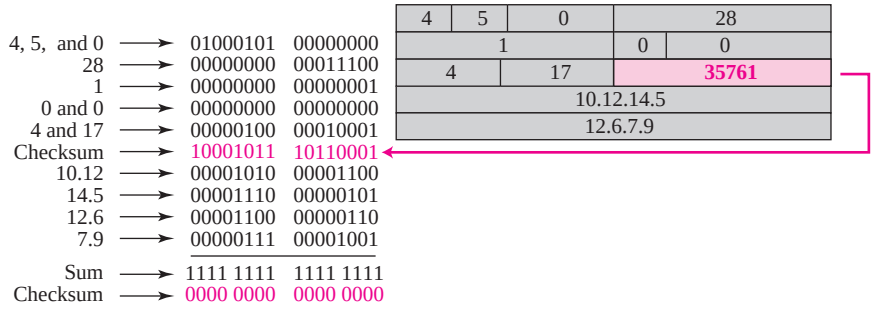
**Figure 7.24** *Example of checksum calculation at the sender*




### Example 7.18

Figure 7.25 shows the checking of checksum calculation at the receiver site (or intermediate router) assuming that no errors occurred in the header. The header is divided into 16-bit sections. All the sections are added and the sum is complemented. Since the result is 16 0s, the packet is accepted.

**Figure 7.25** *Example of checksum calculation at the receiver*




## 7.6 IP OVER ATM

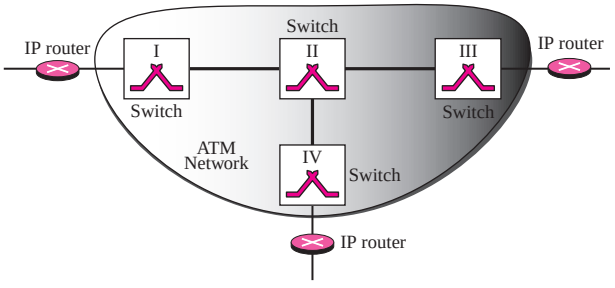
In the previous sections, we assumed that the underlying networks over which the IP datagrams are moving are either LANs or point-to-point WANs. In this section, we want to see how an IP datagram is moving through a switched WAN such as an ATM. We will see that there are similarities as well as differences. The IP packet is encapsulated in cells (not just one). An ATM network has its own definition for the physical address of a device. Binding between an IP address and a physical address is attained through a protocol called ATMARP (discussed in Chapter 8).

**Appendix D gives an algorithm for checksum calculation.**

## ATM WANs

We discussed ATM WANs in Chapter 3. ATM, a cell-switched network, can be a highway for an IP datagram. Figure 7.26 shows how an ATM network can be used in the Internet.

**Figure 7.26** *An ATM WAN in the Internet*




### *AAL Layer*

In Chapter 3, we discussed different AAL layers and their applications. The only AAL used by the Internet is AAL5. It is sometimes called the *simple and efficient adaptation layer* (SEAL). AAL5 assumes that all cells created from one IP datagram belong to a single message. AAL5 therefore provides no addressing, sequencing, or other header information. Instead, only padding and a four-field trailer are added to the IP packet.

AAL5 accepts an IP packet of no more than 65,536 bytes and adds an 8-byte trailer as well as any padding required to ensure that the position of the trailer falls where the receiving equipment expects it (at the last 8 bytes of the last cell). Once the padding and trailer are in place, AAL5 passes the message in 48-byte segments to the ATM layer.

**The AAL layer used by the IP protocol is AAL5.**

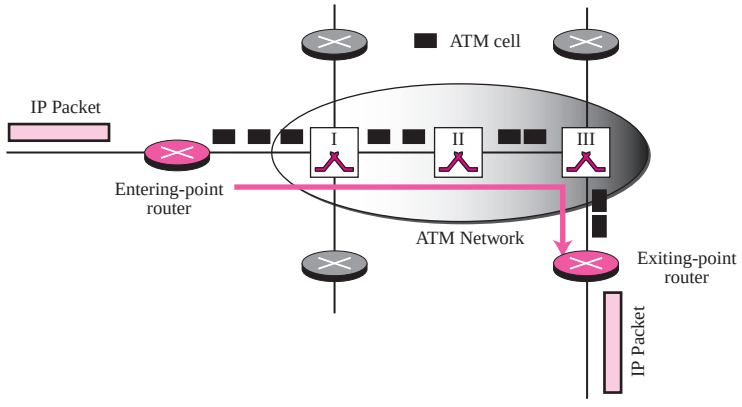
### *Why Use AAL5?*

A question that frequently comes up is why do we use AAL5. Why can't we just encapsulate an IP packet in a cell? The answer is that it is more efficient to use AAL5. If an IP datagram is to be encapsulated in a cell, the data at the IP level must be $53 - 5 - 20 = 27$ bytes because a minimum of 20 bytes is needed for the IP header and 5 bytes is needed for the ATM header. The efficiency is 27/53, or almost 51 percent. By letting an IP datagram span over several cells, we are dividing the IP overhead (20 bytes) among those cells and increasing efficiency.

## Routing the Cells

The ATM network creates a route between two routers. We call these routers entering-point and exiting-point routers. The cells start from the entering-point router and end at the exiting-point router as shown in Figure 7.27.

**Figure 7.27** *Entering-point and exiting-point routers*




### *Addresses*

Routing the cells from one specific entering-point router to one specific exiting-point router requires three types of addressing: IP addresses, physical addresses, and virtual circuit identifiers.

**IP Addresses** Each router connected to the ATM network has an IP address. Later we will see that the addresses may or may not have the same prefix. The IP address defines the router at the IP layer. It does not have anything to do with the ATM network.

**Physical Addresses** Each router (or any other device) connected to the ATM network has also a physical address. The physical address is associated with the ATM network and does not have anything to do with the Internet. The ATM Forum defines 20-byte addresses for ATM networks. Each address must be unique in a network and is defined by the network administrator. The physical addresses in an ATM network play the same role as the MAC addresses in a LAN. The physical addresses are used during connection establishment.

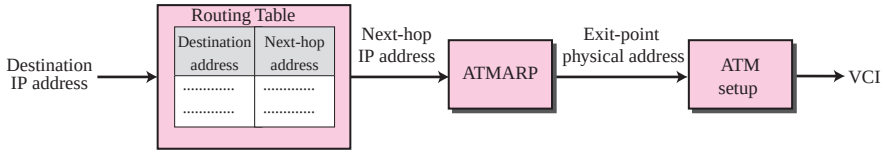
**Virtual Circuit Identifiers** The switches inside the ATM network route the cells based on the virtual circuit identifiers (VPIs and VCIs), as we discussed in Chapter 3. The virtual circuit identifiers are used during data transfer.

### *Address Binding*

An ATM network needs virtual circuit identifiers to route the cells. The IP datagram contains only source and destination IP addresses. Virtual circuit identifiers must be determined from the destination IP address. Figure 7.28 shows how this is done. These are the steps:

1. The **entering-point router** receives an IP datagram. It uses the destination address and its routing table to find the IP address of the next router, the **exiting-point router.** This is exactly the same step followed when a datagram passes through a LAN.

**Figure 7.28** *Address binding in IP over ATM*




2. The entering-point router uses the services of a protocol called ATMARP to find the physical address of the exiting-point router. ATMARP is similar to ARP (discussed in Chapter 8).
3. The virtual circuit identifiers are bound to the physical addresses as discussed in Chapter 3.

## 7.7 SECURITY

The IPv4 protocol, as well as the whole Internet, was started when the Internet users trusted each other. No security was provided for the IPv4 protocol. Today, however, the situation is different; the Internet is not secure any more. Although we discuss network security in general and IP security in particular in Chapters 29 and 30, we give a brief idea about the security issues in IP protocol and the solution.

### Security Issues

There are three security issues that are particularly applicable to the IP protocol: packet sniffing, packet modification, and IP spoofing.

#### *Packet Sniffing*

An intruder may intercept an IP packet and make a copy of it. Packet sniffing is a passive attack, in which the attacker does not change the contents of the packet. This type of attack is very difficult to detect because the sender and the receiver may never know that the packet has been copied. Although packet sniffing cannot be stopped, *encryption* of the packet can make the attacker effort useless. The attacker may still sniff the packet, but it cannot find its contents.

#### *Packet Modification*

The second type of attack is to modify the packet. The attacker intercepts the packet, changes its contents, and sends the new packet to the receiver. The receiver believes that the packet is coming from the original sender. This type of attack can be detected using a *data integrity* mechanism. The receiver before opening and using the contents of the message can use this mechanism to make sure that the packet has not been changed during the transmission.

#### *IP Spoofing*

An attacker can masquerade as somebody else and create an IP packet that carries the source address of another computer. An attacker can send an IP packet to a bank

pretending that it is coming from one of the customers. This type of attack can be prevented using an *origin authentication* mechanism.

### IPSec

The IP packets today can be protected from the previously mentioned attacks using a protocol called IPSec (IP Security). This protocol, which is used in conjunction with the IP protocol, creates a connection-oriented service between two entities in which they can exchange IP packets without worrying about the three attacks discussed before. We will discuss IPSec in detail in Chapter 30, it is enough to mention that IPSec provides the following four services:

#### *Defining Algorithms and Keys*

The two entities that want to create a secure channel between themselves can agree on some available algorithms and keys to be used for security purposes.

#### *Packets Encryption*

The packets exchanged between two parties can be encrypted for privacy using one of the encryption algorithms and a shared key agreed upon in the first step. This makes the packet sniffing attack useless.

#### *Data Integrity*

Data integrity guarantees that the packet is not modified during the transmission. If the received packet does not pass the data integrity test, it is discarded. This prevents the second attack, packet modification, described above.

#### *Origin Authentication*

IPsec can authenticate the origin of the packet to be sure that the packet is not created by an imposter. This can prevent IP spoofing attack as described above.

## 7.8 IP PACKAGE

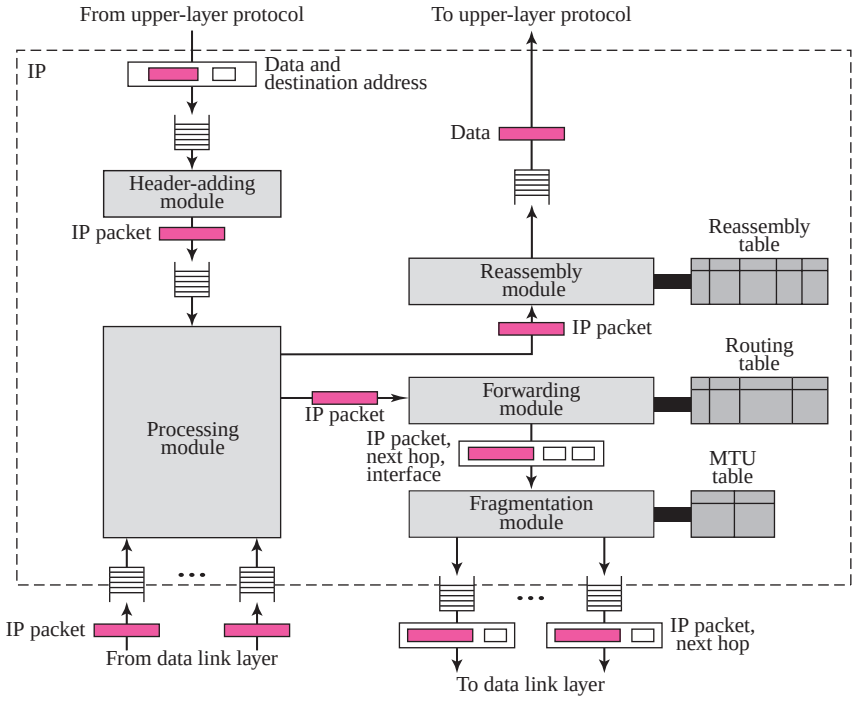
In this section, we present a simplified example of a hypothetical IP package. Our purpose is to show the relationships between the different concepts discussed in this chapter. Figure 7.29 shows eight components and their interactions.

Although IP supports several options, we have omitted option processing in our package to make it easier to understand at this level. In addition, we have sacrificed efficiency for the sake of simplicity.

We can say that the IP package involves eight components: a header-adding module, a processing module, a forwarding module, a fragmentation module, a reassembly module, a routing table, an MTU table, and a reassembly table. In addition, the package includes input and output queues.

The package receives a packet, either from the data link layer or from a higher-level protocol. If the packet comes from an upper-layer protocol, it is delivered to the data link layer for transmission (unless it has a loopback address of 127.X.Y.Z). If the packet comes from the data link layer, it is either delivered to the data link layer for forwarding (in a router) or it is delivered to a higher-layer protocol if the destination IP

**Figure 7.29** *IP components*




address of the packet is the same as the station IP address. Note that we used multiple queues to and from the data link layer because a router is multihomed.

## Header-Adding Module

The **header-adding module** (Table 7.3) receives data from an upper-layer protocol along with the destination IP address. It encapsulates the data in an IP datagram by adding the IP header.

**Table 7.3** *Adding module*

```
1  IP_Adding_Module (data, destination_address)
2  {
3       Encapsulate data in an IP datagram
4       Calculate checksum and insert it in the checksum field
5       Send data to the corresponding queue
6       Return
7  }
```

## Processing Module

The **processing module** (Table 7.4) is the heart of the IP package. In our package, the processing module receives a datagram from an interface or from the header-adding module. It treats both cases the same. A datagram must be processed and routed regardless of where it comes from.

**Table 7.4** *Processing module*

```
IP_Processing_Module (Datagram)
{
      Remove one datagram from one of the input queues.
      If (destination address matches a local address)
      {
          Send the datagram to the reassembly module.
          Return.
      }
      If (machine is a router)
      {
          Decrement TTL.
      }
      If (TTL less than or equal to zero)
      {
          Discard the datagram.
          Send an ICMP error message.
          Return.
      }
      Send the datagram to the forwarding module.
      Return.
}
```

The processing module first checks to see if the datagram has reached its final destination. In this case, the packet is sent to the reassembly module.

If the node is a router, it decrements the time-to-live (TTL) field by one. If this value is less than or equal to zero, the datagram is discarded and an ICMP message (see Chapter 9) is sent to the original sender. If the value of TTL is greater than zero after decrement, the processing module sends the datagram to the forwarding module.

## Queues

Our package uses two types of queues: input queues and output queues. The **input queues** store the datagrams coming from the data link layer or the upper-layer protocols. The **output queues** store the datagrams going to the data link layer or the upper-layer protocols. The processing module dequeues (removes) the datagrams from the input queues. The fragmentation and reassembly modules enqueue (add) the datagrams into the output queues.

## Routing Table

We discussed the routing table in Chapter 6. The routing table is used by the forwarding module to determine the next-hop address of the packet.

## Forwarding Module

We discussed the **forwarding module** in Chapter 6. The forwarding module receives an IP packet from the processing module. If the packet is to be forwarded, it is passed to this module. The module finds the IP address of the next station along with the interface number to which the packet should be sent. It then sends the packet with this information to the fragmentation module.

## MTU Table

The MTU table is used by the fragmentation module to find the **maximum transfer unit (MTU)** of a particular interface. It can have only two columns: interface and MTU.

## Fragmentation Module

In our package, the **fragmentation module** (Table 7.5) receives an IP datagram from the forwarding module. The forwarding module gives the IP datagram, the IP address of the next station (either the final destination in a direct delivery or the next router in an indirect delivery), and the interface number through which the datagram is sent out.

The fragmentation module consults the MTU table to find the MTU for the specific interface number. If the length of the datagram is larger than the MTU, the fragmentation module fragments the datagram, adds a header to each fragment, and sends them to the ARP package (see Chapter 8) for address resolution and delivery.

**Table 7.5** *Fragmentation module*

```
IP_Fragmentation_Module (datagram)
{
     Extract the size of datagram
     If (size > MTU of the corresponding network)
     {
          If (D bit is set)
          {
               Discard datagram
               Send an ICMP error message
               return
          }
          Else
          {
               Calculate maximum size
               Divide the segment into fragments
               Add header to each fragment
               Add required options to each fragment
```

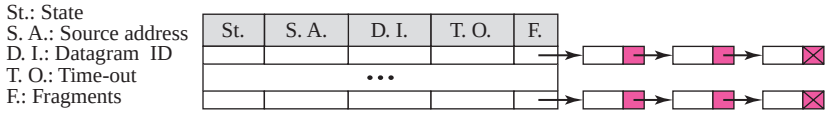
**Table 7.5** *Fragmentation module (continued)*

```
                Send fragment
                return
            }
        }
        Else
        {
            Send the datagram
        }
        Return.
}
```

## Reassembly Table

The **reassembly table** is used by the reassembly module. In our package, the reassembly table has five fields: state, source IP address, datagram ID, time-out, and fragments (see Figure 7.30).

**Figure 7.30** *Reassembly table*




The value of the state field can be either FREE or IN-USE. The IP address field defines the source IP address of the datagram. The datagram ID is a number that uniquely defines a datagram and all of the fragments belonging to that datagram. The time-out is a predetermined amount of time in which all fragments must arrive. Finally, the fragments field is a pointer to a linked list of fragments.

## Reassembly Module

The **reassembly module** (Table 7.6) receives, from the processing module, those datagram fragments that have arrived at their final destinations. In our package, the reassembly module treats an unfragmented datagram as a fragment belonging to a datagram with only one fragment.

Because the IP protocol is a connectionless protocol, there is no guarantee that the fragments arrive in order. Besides, the fragments from one datagram can be intermixed with fragments from another datagram. To keep track of these situations, the module uses a reassembly table with associated linked lists, as we described earlier.

The job of the reassembly module is to find the datagram to which a fragment belongs, to order the fragments belonging to the same datagram, and reassemble all fragments of a datagram when all have arrived. If the established time-out has expired and any fragment is missing, the module discards the fragments.

**Table 7.6** *Reassembly module*

```
IP_Reassembly_Module (datagram)
{
     If (offset value = 0 AND M = 0)
     {
          Send datagram to the appropriate queue
          Return
     }
     Search the reassembly table for the entry
     If (entry not found)
     {
          Create a new entry
     }
     Insert datagram into the linked list
     If (all fragments have arrived)
     {
          Reassemble the fragment
          Deliver the fragment to upper-layer protocol
          return
     }
     Else
     {
          If (time-out expired)
          {
               Discard all fragments
               Send an ICMP error message
          }
     }
     Return.
}
```

## 7.9 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], [Kur & Ros 08], and [Gar & Vid 04].

### RFCs

Several RFCs discuss the IPv4 protocol including: RFC 791, RFC 815, RFC 894, RFC 1122, RFC 2474, and RFC 2475.

## 7.10 KEY TERMS

best-effort delivery
checksum
codepoint
datagram
destination address
differentiated services
end-of-option option
entering-point router
exiting-point router
forwarding module
fragmentation
fragmentation module
fragmentation offset
header length
header-adding module
input queue
Internet Protocol (IP)
length field
loose-source-route option
maximum transfer unit (MTU)
no-operation option
output queue
ping
pointer field
precedence
processing module
reassembly module
reassembly table
record-route option
service type
source address
strict-source-route option
time to live
timestamp option
traceroute
type field
type of service (TOS)
value field

## 7.11 SUMMARY

- IP is an unreliable connectionless protocol responsible for source-to-destination delivery. Packets in the IP layer are called datagrams.
- The MTU is the maximum number of bytes that a data link protocol can encapsulate. MTUs vary from protocol to protocol. Fragmentation is the division of a datagram into smaller units to accommodate the MTU of a data link protocol.
- The IP datagram header consists of a fixed, 20-byte section and a variable options section with a maximum of 40 bytes. The options section of the IP header is used for network testing and debugging. The six IP options each have a specific function.
- The error detection method used by IP is the checksum. The checksum, however, covers only the header, but not the data. The checksum uses one's complement arithmetic to add equal-size sections of the IP header. The complemented result is stored in the checksum field. The receiver also uses one's complement arithmetic to check the header.

- IP over ATM uses AAL5 layer in an ATM network. An ATM network creates a route between an entering-point router and an exiting-point router. The next-hop address of an IP packet can be mapped to a physical address of an exiting-point router using ATMARP.
- An IP package can consist of the following: a header-adding module, a processing module, a forwarding module, a fragmentation module, a reassembly module, a routing table, an MTU table, and a reassembly table.

## 7.12 PRACTICE SET

### Exercises

1. Which fields of the IP header change from router to router?
2. Calculate the HLEN value if the total length is 1200 bytes, 1176 of which is data from the upper layer.
3. Table 7.3 lists the MTUs for many different protocols. The MTUs range from 296 to 65,535. What would be the advantages of having a large MTU? What would be the advantages of having a small MTU?
4. Given a fragmented datagram with an offset of 120, how can you determine the first and last byte number?
5. An IP datagram must go through router 128.46.10.5. There are no other restrictions on the routers to be visited. Draw the IP options with their values.
6. What is the maximum number of routers that can be recorded if the timestamp option has a flag value of 1? Why?
7. Can the value of the header length in an IP packet be less than 5? When is it exactly 5?
8. The value of HLEN in an IP datagram is 7. How many option bytes are present?
9. The size of the option field of an IP datagram is 20 bytes. What is the value of HLEN? What is the value in binary?
10. The value of the total length field in an IP datagram is 36 and the value of the header length field is 5. How many bytes of data is the packet carrying?
11. A datagram is carrying 1024 bytes of data. If there is no option information, what is the value of the header length field? What is the value of the total length field?
12. A host is sending 100 datagrams to another host. If the identification number of the first datagram is 1024, what is the identification number of the last?
13. An IP datagram arrives with fragmentation offset of 0 and an *M* bit (more fragment bit) of 0. Is this a first fragment, middle fragment, or last fragment?
14. An IP fragment has arrived with an offset value of 100. How many bytes of data were originally sent by the source before the data in this fragment?
15. An IP datagram has arrived with the following information in the header (in hexadecimal):

**45 00 00 54 00 03 00 00 20 06 00 00 7C 4E 03 02 B4 0E 0F 02**

**a.** Are there any options?
**b.** Is the packet fragmented?
**c.** What is the size of the data?
**d.** Is a checksum used?
**e.** How many more routers can the packet travel to?
**f.** What is the identification number of the packet?
**g.** What is the type of service?

**16.** In a datagram, the M bit is zero, the value of HLEN is 5, the value of total length is 200, and the offset value is 200. What is the number of the first byte and number of the last byte in this datagram? Is this the last fragment, the first fragment, or a middle fragment?

## Research Activities

**17.** Use the *ping* utility with the -R option to check the routing of a packet to a destination. Interpret the result.

**18.** Use the *traceroute* utility with the -g option to implement the loose source route option. Choose some routers between the source and destination. Interpret the result and find if all defined routers have been visited.

**19.** Use the *traceroute* utility with the -G option to implement the strict source route option. Choose some routers between the source and destination. Interpret the result and find if all defined routers have been visited and no undefined router visited.

CHAPTER

# 8

# *Address Resolution Protocol (ARP)*

Before the IP protocol can deliver a packet from a source host to the destination host, it needs to know how to deliver it to the next hop first. An IP packet can consult its routing table, as discussed in Chapter 6, to find the IP address of the next hop. But since IP uses the services of the data link layer, it needs to know the physical address of the next hop. This can be done using a protocol, called Address Resolution Protocol (ARP), which we discuss in this section.

## OBJECTIVES

*The chapter has several objectives:*

- To make a distinction between logical address (IP address), which is used at the network layer, and physical address (MAC address), which is used at the data link layer.
- To describe how the mapping of a logical address to a physical address can be static or dynamic.
- To show how the address resolution protocol (ARP) is used to dynamically map a logical address to a physical address.
- To show that the proxy ARP can be used to create a subnetting effect.
- To discuss ATMARP, which maps the IP addresses when the underlying network is an ATM WAN.
- To show that an ARP software package can be made of five components: a cache table, queues, an output module, an input module, and a cache-control module.
- To show the pseudocode for each module used in the ARP software package.

## 8.1 ADDRESS MAPPING

An internet is made of a combination of physical networks connected together by internetworking devices such as routers. A packet starting from a source host may pass through several different physical networks before finally reaching the destination host.

The hosts and routers are recognized at the network level by their logical addresses. A logical address is an internetwork address. Its jurisdiction is universal. A logical address is unique universally. It is called a *logical* address because it is usually implemented in software. Every protocol that deals with interconnecting networks requires logical addresses. The logical addresses in the TCP/IP protocol suite are called **IP addresses** and are 32 bits long.

However, packets pass through physical networks to reach these hosts and routers. At the physical level, the hosts and routers are recognized by their physical addresses. A **physical address** is a local address. Its jurisdiction is a local network. It should be unique locally, but not necessarily universally. It is called a *physical* address because it is usually (but not always) implemented in hardware. Examples of physical addresses are 48-bit MAC addresses in the Ethernet protocol, which are imprinted on the NIC installed in the host or router.

The physical address and the logical address are two different identifiers. We need both of them because a physical network such as Ethernet can have two different protocols at the network layer such as IP and IPX (Novell) at the same time. Likewise, a packet at a network layer such as IP may pass through different physical networks such as Ethernet and LocalTalk (Apple).

This means that delivery of a packet to a host or a router requires two levels of addressing: logical and physical. We need to be able to map a logical address to its corresponding physical address and vice versa. These can be done using either static or dynamic mapping.

### Static Mapping

**Static mapping** means creating a table that associates a logical address with a physical address. This table is stored in each machine on the network. Each machine that knows, for example, the IP address of another machine but not its physical address can look it up in the table. This has some limitations because physical addresses may change in the following ways:

1. A machine could change its NIC, resulting in a new physical address.
2. In some LANs, such as LocalTalk, the physical address changes every time the computer is turned on.
3. A mobile computer can move from one physical network to another, resulting in a change in its physical address.

To implement these changes, a static mapping table must be updated periodically. This overhead could affect network performance.

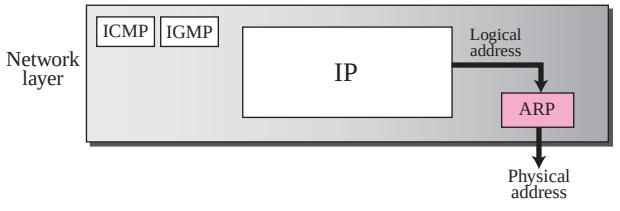
### Dynamic Mapping

In **dynamic mapping**, each time a machine knows the logical address of another machine, it can use a protocol to find the physical address. Two protocols have been designed to perform dynamic mapping: **Address Resolution Protocol (ARP)** and **Reverse Address Resolution Protocol (RARP)**. ARP maps a logical address to a physical address; RARP maps a physical address to a logical address. Since RARP is replaced with another protocol and therefore deprecated, we discuss only ARP protocol in this chapter.

## 8.2 THE ARP PROTOCOL

Anytime a host or a router has an IP datagram to send to another host or router, it has the logical (IP) address of the receiver. But the IP datagram must be encapsulated in a frame to be able to pass through the physical network. This means that the sender needs the physical address of the receiver. A mapping corresponds a logical address to a physical address. Figure 8.1 shows the position of the ARP in the TCP/IP protocol suite. ARP accepts a logical address from the IP protocol, maps the address to the corresponding physical address and pass it to the data link layer.

**Figure 8.1** *Position of ARP in TCP/IP protocol suite*



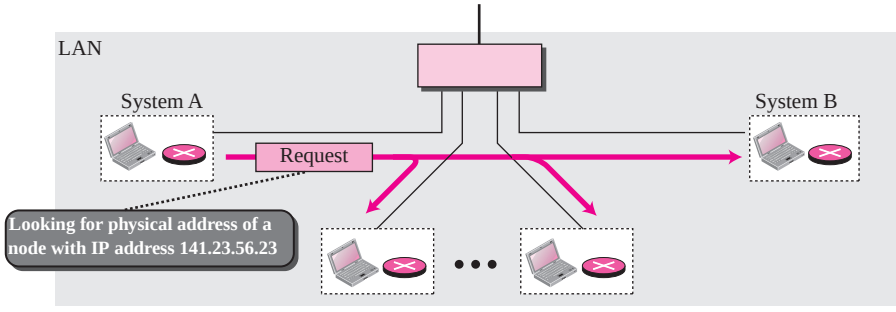

ARP associates an IP address with its physical address. On a typical physical network, such as a LAN, each device on a link is identified by a physical or station address that is usually imprinted on the NIC.

Anytime a host, or a router, needs to find the physical address of another host or router on its network, it sends an ARP query packet. The packet includes the physical and IP addresses of the sender and the IP address of the receiver. Because the sender does not know the physical address of the receiver, the query is broadcast over the network (see Figure 8.2).

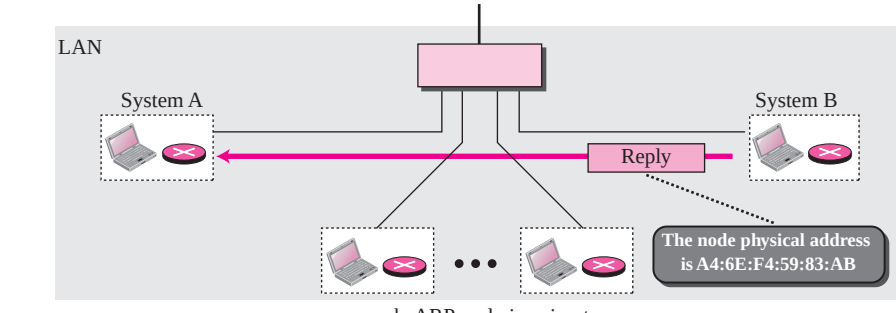
Every host or router on the network receives and processes the ARP query packet, but only the intended recipient recognizes its IP address and sends back an ARP response packet. The response packet contains the recipient's IP and physical addresses. The packet is unicast directly to the inquirer using the physical address received in the query packet.

In Figure 8.2a, the system on the left (A) has a packet that needs to be delivered to another system (B) with IP address 141.23.56.23. System A needs to pass the packet to its data link layer for the actual delivery, but it does not know the physical address of

**Figure 8.2** *ARP operation*




a. ARP request is multicast




b. ARP reply is unicast

the recipient. It uses the services of ARP by asking the ARP protocol to send a broadcast ARP request packet to ask for the physical address of a system with an IP address of 141.23.56.23.

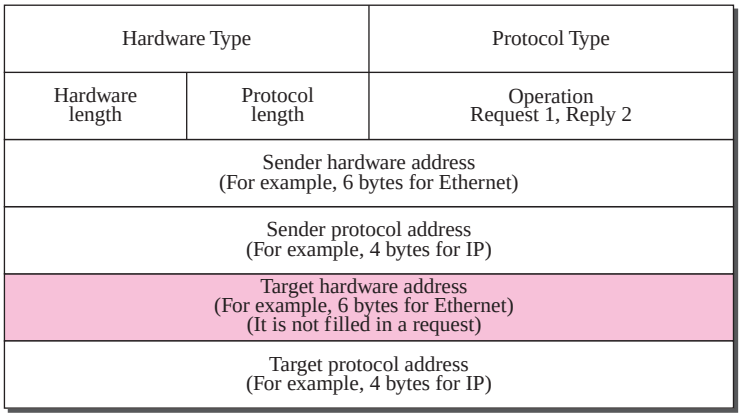
This packet is received by every system on the physical network, but only system B will answer it, as shown in Figure 8.2b. System B sends an ARP reply packet that includes its physical address. Now system A can send all the packets it has for this destination using the physical address it received.

## Packet Format

Figure 8.3 shows the format of an ARP packet. The fields are as follows:

- **Hardware type.** This is a 16-bit field defining the type of the network on which ARP is running. Each LAN has been assigned an integer based on its type. For example, Ethernet is given the type 1. ARP can be used on any physical network.
- **Protocol type.** This is a 16-bit field defining the protocol. For example, the value of this field for the IPv4 protocol is $0800_{16}$. ARP can be used with any higher-level protocol.
- **Hardware length.** This is an 8-bit field defining the length of the physical address in bytes. For example, for Ethernet the value is 6.
- **Protocol length.** This is an 8-bit field defining the length of the logical address in bytes. For example, for the IPv4 protocol the value is 4.

**Figure 8.3** *ARP packet*




- **Operation.** This is a 16-bit field defining the type of packet. Two packet types are defined: ARP request (1), ARP reply (2).
- **Sender hardware address.** This is a variable-length field defining the physical address of the sender. For example, for Ethernet this field is 6 bytes long.
- **Sender protocol address.** This is a variable-length field defining the logical (for example, IP) address of the sender. For the IP protocol, this field is 4 bytes long.
- **Target hardware address.** This is a variable-length field defining the physical address of the target. For example, for Ethernet this field is 6 bytes long. For an ARP request message, this field is all 0s because the sender does not know the physical address of the target.
- **Target protocol address.** This is a variable-length field defining the logical (for example, IP) address of the target. For the IPv4 protocol, this field is 4 bytes long.

## Encapsulation

An ARP packet is encapsulated directly into a data link frame. For example, in Figure 8.4 an ARP packet is encapsulated in an Ethernet frame. Note that the type field indicates that the data carried by the frame is an ARP packet.

## Operation

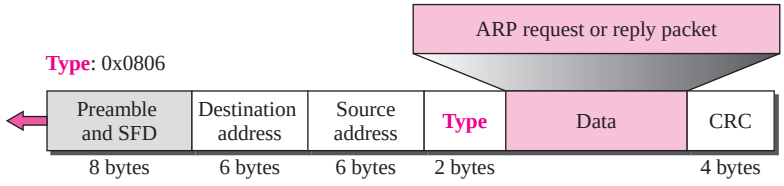
Let us see how ARP functions on a typical internet. First we describe the steps involved. Then we discuss the four cases in which a host or router needs to use ARP.

### *Steps Involved*

These are seven steps involved in an ARP process:

1. The sender knows the IP address of the target. We will see how the sender obtains this shortly.
2. IP asks ARP to create an ARP request message, filling in the sender physical address, the sender IP address, and the target IP address. The target physical address field is filled with 0s.

**Figure 8.4** *Encapsulation of ARP packet*




3. The message is passed to the data link layer where it is encapsulated in a frame using the physical address of the sender as the source address and the physical broadcast address as the destination address.
4. Every host or router receives the frame. Because the frame contains a broadcast destination address, all stations remove the message and pass it to ARP. All machines except the one targeted drop the packet. The target machine recognizes the IP address.
5. The target machine replies with an ARP reply message that contains its physical address. The message is unicast.
6. The sender receives the reply message. It now knows the physical address of the target machine.
7. The IP datagram, which carries data for the target machine, is now encapsulated in a frame and is unicast to the destination.

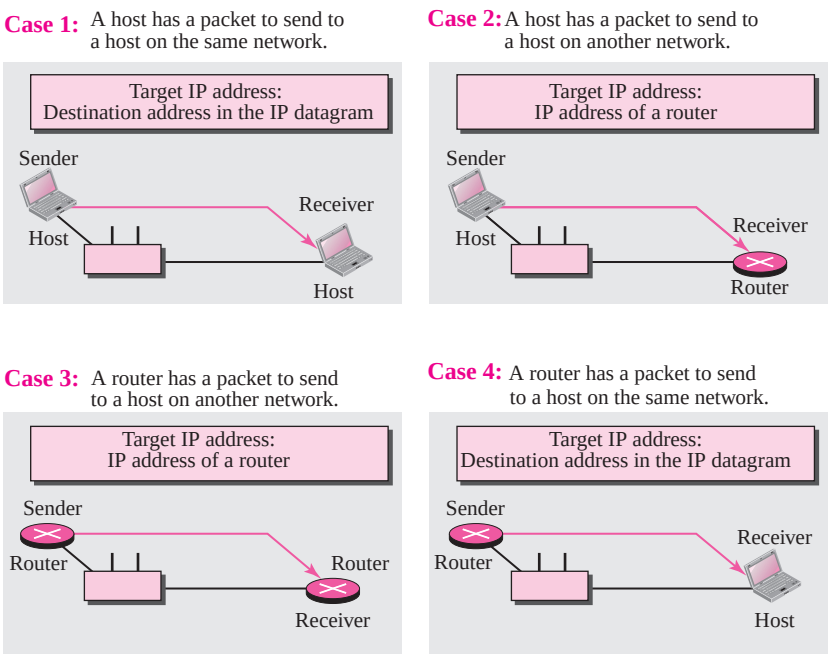
**An ARP request is broadcast; an ARP reply is unicast.**

### *Four Different Cases*

The following are four different cases in which the services of ARP can be used (see Figure 8.5).

- **Case 1:** The sender is a host and wants to send a packet to another host on the same network. In this case, the logical address that must be mapped to a physical address is the destination IP address in the datagram header.
- **Case 2:** The sender is a host and wants to send a packet to another host on another network. In this case, the host looks at its routing table and finds the IP address of the next hop (router) for this destination. If it does not have a routing table, it looks for the IP address of the default router. The IP address of the router becomes the logical address that must be mapped to a physical address.
- **Case 3:** The sender is a router that has received a datagram destined for a host on another network. It checks its routing table and finds the IP address of the next router. The IP address of the next router becomes the logical address that must be mapped to a physical address.
- **Case 4:** The sender is a router that has received a datagram destined for a host in the same network. The destination IP address of the datagram becomes the logical address that must be mapped to a physical address.

**Figure 8.5** *Four cases using ARP*




### Example 8.1

A host with IP address 130.23.43.20 and physical address B2:34:55:10:22:10 has a packet to send to another host with IP address 130.23.43.25 and physical address A4:6E:F4:59:83:AB (which is unknown to the first host). The two hosts are on the same Ethernet network. Show the ARP request and reply packets encapsulated in Ethernet frames.

**Solution**

Figure 8.6 shows the ARP request and reply packets. Note that the ARP data field in this case is 28 bytes, and that the individual addresses do not fit in the 4-byte boundary. That is why we do not show the regular 4-byte boundaries for these addresses. Also note that the IP addresses are shown in hexadecimal. For information on binary or hexadecimal notation see Appendix B.

## Proxy ARP

A technique called *proxy* ARP is used to create a subnetting effect. A **proxy ARP** is an ARP that acts on behalf of a set of hosts. Whenever a router running a proxy ARP receives an ARP request looking for the IP address of one of these hosts, the router sends an ARP reply announcing its own hardware (physical) address. After the router receives the actual IP packet, it sends the packet to the appropriate host or router.

Let us give an example. In Figure 8.7 the ARP installed on the right-hand host will answer only to an ARP request with a target IP address of 141.23.56.23.

**Figure 8.6** *Example 8.1*

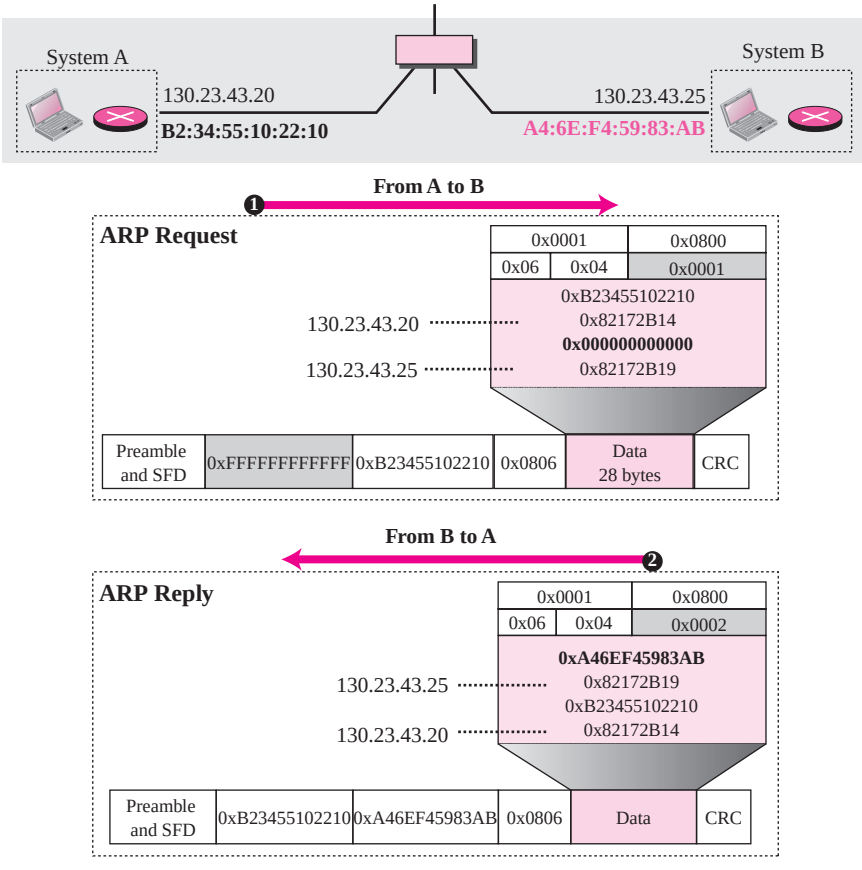
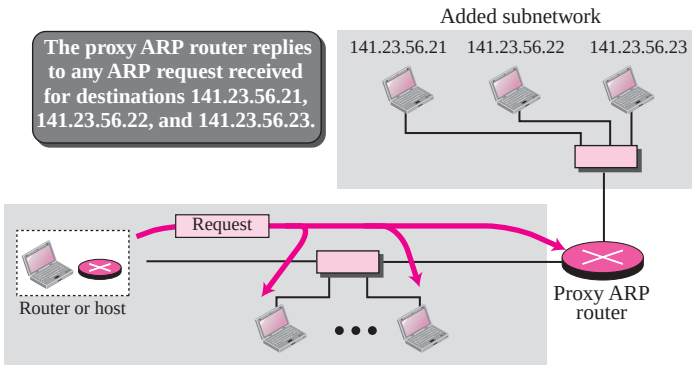



**Figure 8.7** *Proxy ARP*



However, the administrator may need to create a subnet without changing the whole system to recognize subnetted addresses. One solution is to add a router running a proxy ARP. In this case, the router acts on behalf of all of the hosts installed on the subnet. When it receives an ARP request with a target IP address that matches the address of one of its protégés (141.23.56.21, 141.23.56.22, and 141.23.56.23), it sends an ARP reply and announces its hardware address as the target hardware address. When the router receives the IP packet, it sends the packet to the appropriate host.

## 8.3 ATMARP

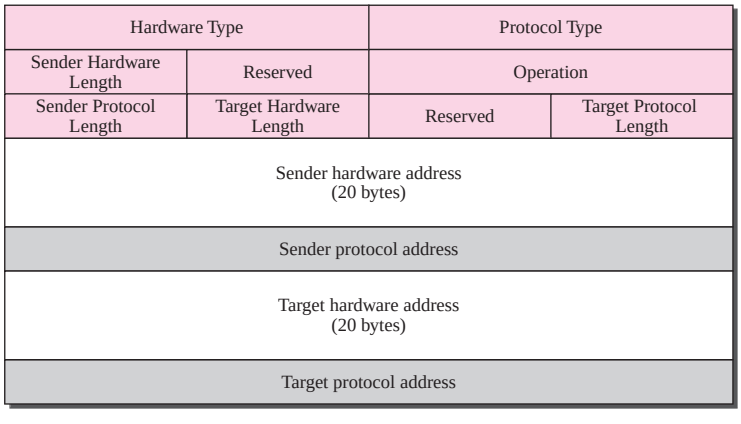
We discussed IP over ATM in Chapter 7. When IP packet are moving through an ATM WAN, a mechanism protocol is needed to find (map) the physical address of the exiting-point router in the ATM WAN given the IP address of the router. This is the same task performed by ARP on a LAN. However, there is a difference between a LAN and an ATM network. A LAN is a broadcast network (at the data link layer); ARP uses the broadcasting capability of a LAN to send (broadcast) an ARP request. An ATM network is not a broadcast network; another solution is needed to handle the task.

### Packet Format

The format of an **ATMARP** packet, which is similar to the ARP packet, is shown in Figure 8.8. The fields are as follows:

- **Hardware type (HTYPE).** The 16-bit HTYPE field defines the type of the physical network. Its value is $0013_{16}$ for an ATM network.
- **Protocol type (PTYPE).** The 16-bit PTYPE field defines the type of the protocol. For IPv4 protocol the value is $0800_{16}$.
- **Sender hardware length (SHLEN).** The 8-bit SHLEN field defines the length of the sender's physical address in bytes. For an ATM network the value is 20. Note

**Figure 8.8** *ATMARP packet*



that if the binding is done across an ATM network and two levels of hardware addressing are necessary, the neighboring 8-bit **reserved field** is used to define the length of the second address.

- **Operation (OPER).** The 16-bit OPER field defines the type of the packet. Five packet types are defined as shown in Table 8.1.

**Table 8.1** *OPER field*

| *Message* | *OPER value* |
|---|---|
| Request | 1 |
| Reply | 2 |
| Inverse Request | 8 |
| Inverse Reply | 9 |
| NACK | 10 |

- **Sender protocol length (SPLEN).** The 8-bit SPLEN field defines the length of the address in bytes. For IPv4 the value is 4 bytes.
- **Target hardware length (TLEN).** The 8-bit TLEN field defines the length of the receiver's physical address in bytes. For an ATM network the value is 20. Note that if the binding is done across an ATM network and two levels of hardware addressing are necessary, the neighboring 8-bit reserved field is used to define the length of the second address.
- **Target protocol length (TPLEN).** The 8-bit TPLEN field defines the length of the address in bytes. For IPv4 the value is 4 bytes.
- **Sender hardware address (SHA).** The variable-length SHA field defines the physical address of the sender. For ATM networks defined by the ATM Forum, the length is 20 bytes.
- **Sender protocol address (SPA).** The variable-length SPA field defines the address of the sender. For IPv4 the length is 4 bytes.
- **Target hardware address (THA).** The variable-length THA field defines the physical address of the receiver. For ATM networks defined by the ATM Forum, the length is 20 bytes. This field is left empty for request messages and filled in for reply and NACK messages.
- **Target protocol address (TPA).** The variable-length TPA field defines the address of the receiver. For IPv4 the length is 4 bytes.

## ATMARP Operation

There are two methods to connect two routers on an ATM network: through a permanent virtual circuit (PVC) or through a switched virtual circuit (SVC). The operation of ATMARP depends on the connection method.

### *PVC Connection*

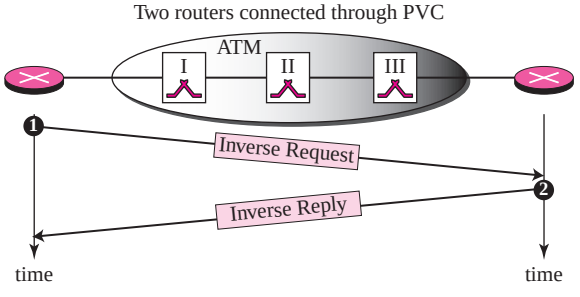
A permanent virtual circuit (PVC) connection is established between two end points by the network provider. The VPIs and VCIs are defined for the permanent connections and the values are entered in a table for each switch.

If a permanent virtual circuit is established between two routers, there is no need for an ATMARP server. However, the routers must be able to bind a physical address to an IP address. The **inverse request message** and **inverse reply message** can be used for the binding. When a PVC is established for a router, the router sends an inverse request message. The router at the other end of the connection receives the message (which contains the physical and IP address of the sender) and sends back an inverse reply message (which contains its own physical and IP address).

After the exchange, both routers add a table entry that maps the physical addresses to the PVC. Now, when a router receives an IP datagram, the table provides information so that the router can encapsulate the datagram using the virtual circuit identifier. Figure 8.9 shows the exchange of messages between two routers.

**The inverse request and inverse reply messages can bind the physical address to an IP address in a PVC situation.**

**Figure 8.9** *Binding with PVC*



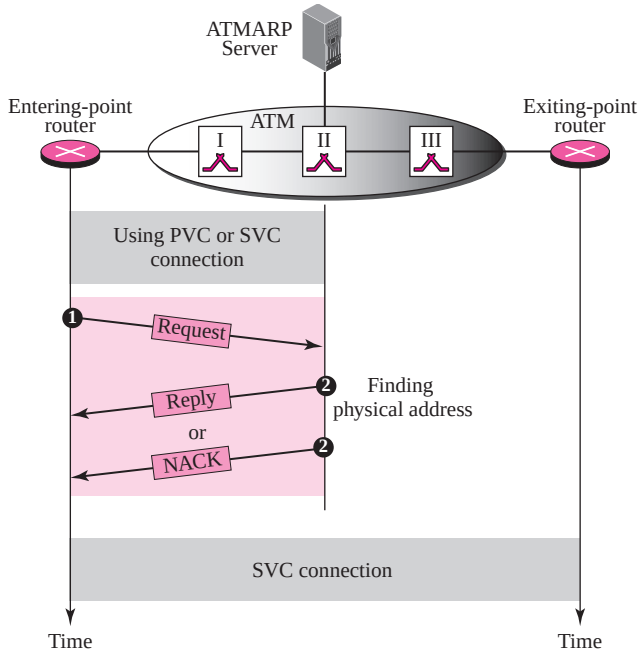

### *SVC Connection*

In a switched virtual circuit (SVC) connection, each time a router wants to make a connection with another router (or any computer), a new virtual circuit must be established. However, the virtual circuit can be created only if the entering-point router knows the physical address of the exiting-point router (ATM does not recognize IP addresses).

To map the IP addresses to physical addresses, each router runs a client ATMARP program, but only one computer runs an ATMARP server program. To understand the difference between ARP and ATMARP, remember that ARP operates on a LAN, which is a broadcast network. An ARP client can broadcast an ARP request message and each router on the network will receive it; only the target router will respond. ATM is a nonbroadcast network; an ATMARP request cannot reach all routers connected to the network.

The process of establishing a virtual connection requires three steps: connecting to the server, receiving the physical address, and establishing the connection. Figure 8.10 shows the steps.

**Figure 8.10** *Binding with ATMARP*




**Connecting to the Server** Normally, there is a permanent virtual circuit established between each router and the server. If there is no PVC connection between the router and the server, the server must at least know the physical address of the router to create an SVC connection just for exchanging ATMARP request and reply messages.

**Receiving the Physical Address** When there is a connection between the entering-point router and the server, the router sends an *ATMARP request* to the server. The server sends back an *ATMARP reply* if the physical address can be found or an *ATMARP NACK* otherwise. If the entering-point router receives a NACK, the datagram is dropped.

**Establishing Virtual Circuits** After the entering-point router receives the physical address of the exiting-point router, it can request an SVC between itself and the exiting-point router. The ATM network uses the two physical addresses to set up a virtual circuit which lasts until the entering-point router asks for disconnection. In this step, each switch inside the network adds an entry to its tables to enable them to route the cells carrying the IP datagram.

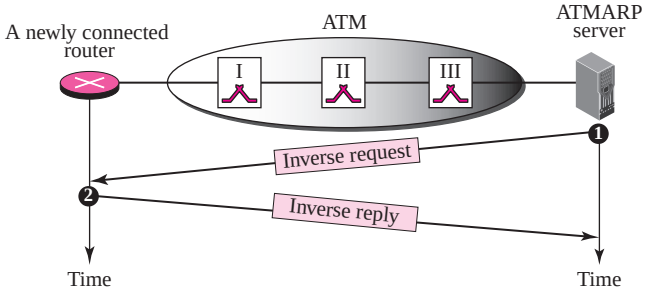
**The request and reply message can be used to bind a physical address to an IP address in an SVC situation.**

### *Building the Table*

How does the ATM server build its mapping table? This is also done through the use of ATMARP and the two inverse messages (inverse request and inverse reply). When a router is connected to an ATM network for the first time and a permanent virtual connection is established between the router and the server, the server sends an inverse request message to the router. The router sends back an inverse reply message, which includes its IP address and physical address. Using these two addresses, the server creates an entry in its routing table to be used if the router becomes an exiting-point router in the future. Figure 8.11 shows the inverse operation of ATMARP.

**The inverse request and inverse reply can also be used to build the server's mapping table.**

**Figure 8.11** *Building a table*




## Logical IP Subnet (LIS)

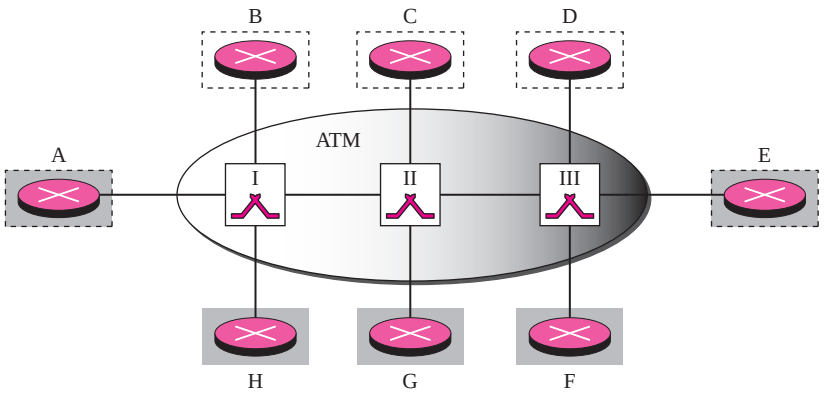
Before we leave the subject of IP over ATM, we need to discuss a concept called **logical IP subnet (LIS).** For the same reason that a large LAN can be divided into several subnets, an ATM network can be divided into logical (not physical) subnetworks. This facilitates the operation of ATMARP and other protocols (such as IGMP) that need to simulate broadcasting on an ATM network.

Routers connected to an ATM network can belong to one or more logical subnets, as shown in Figure 8.12. In the figure, routers B, C, and D belong to one logical subnet (shown by broken-line boxes); routers F, G, and H belong to another logical subnet (shown by shaded boxes). Routers A and E belong to both logical subnets. A router can communicate and send IP packets directly to a router in the same subnet; however, if it needs to send a packet to a router that belongs to another subnet, the packet must first go to a router that belongs to both subnets. For example, router B can send a packet directly to routers C and D. But a packet from B to F must first pass through A or E.

Note that routers belonging to the same logical subnet share the same prefix and subnet mask. The prefix for routers in different subnets is different.

To use ATMARP, there must be a different ATMARP server in each subnet. For example, in the above figure, we need two ATMARP servers, one for each subnet.

**Figure 8.12** *LIS*




**LIS allows an ATM network to be divided into several logical subnets. To use ATMARP, we need a separate server for each subnet.**

## 8.4 ARP PACKAGE

In this section, we give an example of a simplified ARP software package. The purpose is to show the components of a hypothetical ARP package and the relationships between the components. Figure 8.13 shows these components and their interactions.

We can say that this ARP package involves five components: a **cache table,** queues, an output module, an input module, and a cache-control module. The package receives an IP datagram that needs to be encapsulated in a frame that needs the destination physical (hardware) address. If the ARP package finds this address, it delivers the IP packet and the physical address to the data link layer for transmission.

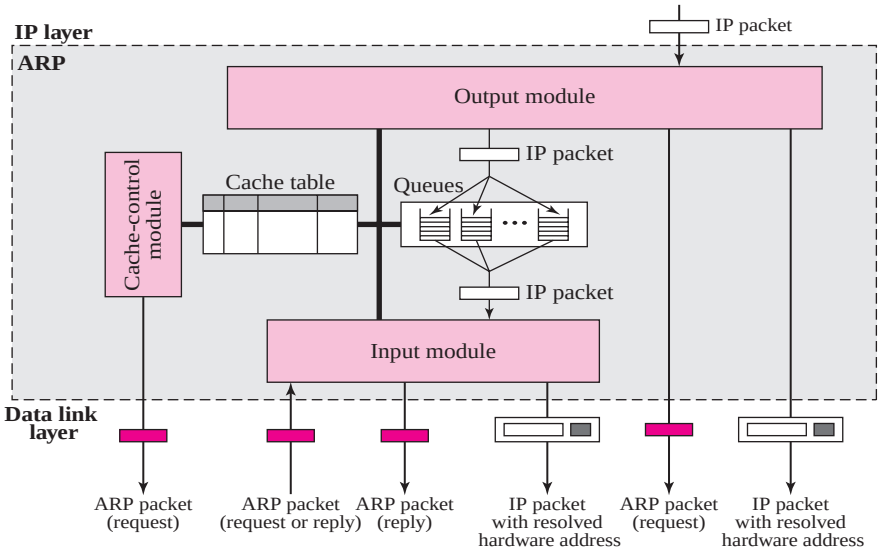
### Cache Table

A sender usually has more than one IP datagram to send to the same destination. It is inefficient to use the ARP protocol for each datagram destined for the same host or router. The solution is the cache table. When a host or router receives the corresponding physical address for an IP datagram, the address can be saved in the cache table. This address can be used for the datagrams destined for the same receiver within the next few minutes. However, as space in the cache table is very limited, mappings in the cache are not retained for an unlimited time.

The cache table is implemented as an array of entries. In our package, each entry contains the following fields:

- **State.** This column shows the state of the entry. It can have one of three values: *FREE*, *PENDING*, or *RESOLVED*. The FREE state means that the time-to-live for

**Figure 8.13** *ARP components*




this entry has expired. The space can be used for a new entry. The PENDING state means a request for this entry has been sent, but the reply has not yet been received. The RESOLVED state means that the entry is complete. The entry now has the physical (hardware) address of the destination. The packets waiting to be sent to this destination can use the information in this entry.

- **Hardware type.** This column is the same as the corresponding field in the ARP packet.
- **Protocol type.** This column is the same as the corresponding field in the ARP packet.
- **Hardware length.** This column is the same as the corresponding field in the ARP packet.
- **Protocol length.** This column is the same as the corresponding field in the ARP packet.
- **Interface number.** A router (or a multihomed host) can be connected to different networks, each with a different interface number. Each network can have different hardware and protocol types.
- **Queue number.** ARP uses numbered queues to enqueue the packets waiting for address resolution. Packets for the same destination are usually enqueued in the same queue.
- **Attempts.** This column shows the number of times an ARP request is sent out for this entry.
- **Time-out.** This column shows the lifetime of an entry in seconds.
- **Hardware address.** This column shows the destination hardware address. It remains empty until resolved by an ARP reply.
- **Protocol address.** This column shows the destination IP address.

## Queues

Our ARP package maintains a set of queues, one for each destination, to hold the IP packets while ARP tries to resolve the hardware address. The output module sends unresolved packets into the corresponding **queue.** The input module removes a packet from a queue and sends it, with the resolved physical address, to the data link layer for transmission.

## Output Module

Table 8.2 shows the output module in pseudocode.

**Table 8.2** *Output Module*

```
ARP_Output_Module ( )
{
  Sleep until an IP packet is received from IP software.
  Check cache table for an entry corresponding to the
       destination of IP packet.
  If (entry is found)
  {
      If (the state is RESOLVED)
      {
        Extract the value of the hardware address from the entry.
        Send the packet and the hardware address to data
             link layer.
        Return
      } // end if
      If (the state is PENDING)
      {
        Enqueue the packet to the corresponding queue.
        Return
     }//end if
  }//end if
  If (entry is not found)
  {
     Create a cache entry with state set to PENDING and
          ATTEMPTS set to 1.
     Create a queue.
     Enqueue the packet.
     Send an ARP request.
     Return
  }//end if
} //end module
```

The output module waits for an IP packet from the IP software. The output module checks the cache table to find an entry corresponding to the destination IP address of

this packet. The destination IP address of the IP packet must match the protocol address of the entry.

If the entry is found and the state of the entry is RESOLVED, the packet along with the destination hardware address is passed to the data link layer for transmission.

If the entry is found and the state of the entry is PENDING, the packet waits until the destination hardware address is found. Because the state is PENDING, there is a queue already created for this destination. The module sends the packet to this queue.

If no entry is found, the module creates a queue and enqueues the packet. A new entry with the state of PENDING is created for this destination and the value of the ATTEMPTS field is set to 1. An ARP request packet is then broadcast.

## Input Module

Table 8.3 shows the input module in pseudocode.

**Table 8.3** *Input Module*

```
1  ARP_Input_Module ( )
2  {
3     Sleep until an ARP packet (request or reply) arrives.
4     Check the cache table to find the corresponding entry.
5     If (found)
6     {
7        Update the entry.
8        If (the state is PENDING)
9        {
10          While (the queue is not empty)
11          {
12             Dequeue one packet.
13             Send the packet and the hardware address.
14          }//end if
15       }//end if
16    }//end if
17    If (not found)
18    {
19       Create an entry.
20       Add the entry to the table.
21    }//end if
22    If (the packet is a request)
23    {
24       Send an ARP reply.
25    }//end if
26    Return
27 }//end module
```

The input module waits until an ARP packet (request or reply) arrives. The input module checks the cache table to find an entry corresponding to this ARP packet. The target protocol address should match the protocol address of the entry.

If the entry is found and the state of the entry is PENDING, the module updates the entry by copying the target hardware address in the packet to the hardware address field of the entry and changing the state to RESOLVED. The module also sets the value of the TIME-OUT for this entry. It then dequeues the packets from the corresponding queue, one by one, and delivers them along with the hardware address to the data link layer for transmission.

If the entry is found and the state is RESOLVED, the module still updates the entry. This is because the target hardware address could have been changed. The value of the TIME-OUT field is also reset.

If the entry is not found, the module creates a new entry and adds it to the table. The protocol requires that any information received is added to the table for future use. The state is set to RESOLVED and TIME-OUT is set.

Now the module checks to see if the arrived ARP packet is a request. If it is, the module immediately creates an ARP reply message and sends it to the sender. The ARP reply packet is created by changing the value of the operation field from request to reply and filling in the target hardware address.

## Cache-Control Module

The **cache-control module** is responsible for maintaining the cache table. It periodically (for example, every 5 s) checks the cache table, entry by entry. If the state of the entry is FREE, it continues to the next entry. If the state is PENDING, the module increments the value of the attempts field by 1. It then checks the value of the attempts field. If this value is greater than the maximum number of attempts allowed, the state is changed to FREE and the corresponding queue is destroyed. However, if the number of attempts is less than the maximum, the module creates and sends another ARP request.

If the state of the entry is RESOLVED, the module decrements the value of the time-out field by the amount of time elapsed since the last check. If this value is less than or equal to zero, the state is changed to FREE and the queue is destroyed. Table 8.4 shows the cache-control module in pseudocode.

**Table 8.4** *Cache-Control Module*

```
ARP_Cache_Control_Module ( )
{
    Sleep until the periodic timer matures.
    Repeat for every entry in the cache table
    {
       If (the state is FREE)
       {
          Continue.
       }//end if
        If (the state is PENDING)
        {
```

**Table 8.4** *Cache-Control Module (continued)*

```
12          Increment the value of attempts by 1.
13          If (attempts greater than maximum)
14          {
15               Change the state to FREE.
16               Destroy the corresponding queue.
17          }// end if
18          else
19          {
20             Send an ARP request.
21           }//end else
22           continue.
23         }//end if
24         If (the state is RESOLVED)
25         {
26             Decrement the value of time-out.
27             If (time-out less than or equal 0)
28             {
29                Change the state to FREE.
30                Destroy the corresponding queue.
31            }//end if
32         }//end if
33       }//end repeat
34       Return.
35    }//end module
```

## More Examples

In this section we show some examples of the ARP operation and the changes in the cache table. Table 8.5 shows some of the cache table fields at the start of our examples.

**Table 8.5** *Original cache table used for examples*

| *State* | *Queue* | *Attempt* | *Time-Out* | *Protocol Addr.* | *Hardware Addr.* |
|---|---|---|---|---|---|
| R | 5 | | 900 | 180.3.6.1 | ACAE32457342 |
| P | 2 | 2 | | 129.34.4.8 | |
| P | 14 | 5 | | 201.11.56.7 | |
| R | 8 | | 450 | 114.5.7.89 | 457342ACAE32 |
| P | 12 | 1 | | 220.55.5.7 | |
| F | | | | | |
| R | 9 | | 60 | 19.1.7.82 | 4573E3242ACA |
| P | 18 | 3 | | 188.11.8.71 | |

### Example 8.2

The ARP output module receives an IP datagram (from the IP layer) with the destination address 114.5.7.89. It checks the cache table and finds that an entry exists for this destination with the RESOLVED state (R in the table). It extracts the hardware address, which is 457342ACAE32, and sends the packet and the address to the data link layer for transmission. The cache table remains the same.

### Example 8.3

Twenty seconds later, the ARP output module receives an IP datagram (from the IP layer) with the destination address 116.1.7.22. It checks the cache table and does not find this destination in the table. The module adds an entry to the table with the state PENDING and the Attempt value 1. It creates a new queue for this destination and enqueues the packet. It then sends an ARP request to the data link layer for this destination. The new cache table is shown in Table 8.6.

**Table 8.6** *Updated cache table for Example 8.3*

| *State* | *Queue* | *Attempt* | *Time-Out* | *Protocol Addr.* | *Hardware Addr.* |
|---|---|---|---|---|---|
| R | 5 | | 900 | 180.3.6.1 | ACAE32457342 |
| P | 2 | 2 | | 129.34.4.8 | |
| P | 14 | 5 | | 201.11.56.7 | |
| R | 8 | | 450 | 114.5.7.89 | 457342ACAE32 |
| P | 12 | 1 | | 220.55.5.7 | |
| P | 23 | 1 | | 116.1.7.22 | |
| R | 9 | | 60 | 19.1.7.82 | 4573E3242ACA |
| P | 18 | 3 | | 188.11.8.71 | |

### Example 8.4

Fifteen seconds later, the ARP input module receives an ARP packet with target protocol (IP) address 188.11.8.71. The module checks the table and finds this address. It changes the state of the entry to RESOLVED and sets the time-out value to 900. The module then adds the target hardware address (E34573242ACA) to the entry. Now it accesses queue 18 and sends all the packets in this queue, one by one, to the data link layer. The new cache table is shown in Table 8.7.

**Table 8.7** *Updated cache table for Example 8.4*

| *State* | *Queue* | *Attempt* | *Time-Out* | *Protocol Addr.* | *Hardware Addr.* |
|---|---|---|---|---|---|
| R | 5 | | 900 | 180.3.6.1 | ACAE32457342 |
| P | 2 | 2 | | 129.34.4.8 | |
| P | 14 | 5 | | 201.11.56.7 | |
| R | 8 | | 450 | 114.5.7.89 | 457342ACAE32 |
| P | 12 | 1 | | 220.55.5.7 | |
| P | 23 | 1 | | 116.1.7.22 | |
| R | 9 | | 60 | 19.1.7.82 | 4573E3242ACA |
| R | 18 | | 900 | 188.11.8.71 | E34573242ACA |

### Example 8.5

Twenty-five seconds later, the cache-control module updates every entry. The time-out values for the first three resolved entries are decremented by 60. The time-out value for the last

resolved entry is decremented by 25. The state of the next-to-the last entry is changed to FREE because the time-out is zero. For each of the three pending entries, the value of the attempts field is incremented by one. After incrementing, the attempts value for one entry (the one with IP address 201.11.56.7) is more than the maximum; the state is changed to FREE, the queue is deleted, and an ICMP message is sent to the original destination (see Chapter 9). See Table 8.8.

**Table 8.8** *Updated cache table for Example 8.5*

| *State* | *Queue* | *Attempt* | *Time-Out* | *Protocol Addr.* | *Hardware Addr.* |
|---|---|---|---|---|---|
| R | 5 | | 840 | 180.3.6.1 | ACAE32457342 |
| P | 2 | 3 | | 129.34.4.8 | |
| F | | | | | |
| R | 8 | | 390 | 114.5.7.89 | 457342ACAE32 |
| P | 12 | 2 | | 220.55.5.7 | |
| P | 23 | 2 | | 116.1.7.22 | |
| F | | | | | |
| R | 18 | | 875 | 188.11.8.71 | E34573242ACA |

## 8.5 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], and [Ste 94].

### RFCs

Several RFCs in particular discuss ARP including RFC 826, RFC 1029, RFC 1166, and RFC 1981.

## 8.6 KEY TERMS

Address Resolution Protocol (ARP)
cache table
cache-control module
dynamic mapping
encapsulation
inverse reply message
inverse request message
IP addresses
logical IP subnet (LIS)
physical address
proxy ARP
queue
reserved field
Reverse Address Resolution Protocol (RARP)
static mapping

## 8.7 SUMMARY

- Delivery of a packet to a host or router requires two levels of addresses: logical and physical. A logical address identifies a host or router at the network level. TCP/IP calls this logical address an IP address. A physical address identifies a host or router at the physical level.
- Mapping of a logical address to a physical address can be static or dynamic. Static mapping involves a list of logical and physical addresses; maintenance of the list requires high overhead.
- The address resolution protocol (ARP) is a dynamic mapping method that finds a physical address given a logical address. An ARP request is broadcast to all devices on the network. An ARP reply is unicast to the host requesting the mapping.
- In proxy ARP, a router represents a set of hosts. When an ARP request seeks the physical address of any host in this set, the router sends its own physical address. This creates a subnetting effect.
- ATMARP is a protocol used on ATM networks that binds a physical address to an IP address. The ATMARP server's mapping table is built through the use of the inverse request and the inverse reply messages. An ATM network can be divided into logical subnetworks to facilitate ATMARP and other protocol operations.
- The ARP software package consists of five components: a cache table, queues, an output module, an input module, and a cache-control module. The cache table has an array of entries used and updated by ARP messages. A queue contains packets going to the same destination. The output module takes a packet from the IP layer and sends it either to the data link layer or to a queue. The input module uses an ARP packet to update the cache table. The input module can also send an ARP reply. The cache-control module maintains the cache table by updating entry fields.

## 8.8 PRACTICE SET

### Exercises

1. Is the size of the ARP packet fixed? Explain.
2. What is the size of an ARP packet when the protocol is IP and the hardware is Ethernet?
3. What is the size of an Ethernet frame carrying an ARP packet?
4. What is the broadcast address for Ethernet?
5. A router with IP address 125.45.23.12 and Ethernet physical address 23:45:AB:4F:67:CD has received a packet for a host destination with IP address 125.11.78.10 and Ethernet physical address AA:BB:A2:4F:67:CD.
   a. Show the entries in the ARP request packet sent by the router. Assume no subnetting.
   b. Show the entries in the ARP packet sent in response to part a.

**c.** Encapsulate the packet made in part a in a data link frame. Fill in all the fields.

**d.** Encapsulate the packet part b in a data link frame. Fill in all the fields.

**6.** A router with IP address 195.5.2.12 and Ethernet physical address AA:25:AB:1F:67:CD has received a packet for a destination with IP address 185.11.78.10. When the router checks its routing table, it finds out the packet should be delivered to a router with IP address 195.5.2.6 and Ethernet physical address AD:34:5D:4F:67:CD.

**a.** Show the entries in the ARP request packet sent by the router. Assume no subnetting.

**b.** Show the entries in the ARP packet sent in response to part a.

**c.** Encapsulate the packet made in part a in the data link layer. Fill in all the fields.

**d.** Encapsulate the packet made in part b in a data link frame. Fill in all the fields.

**7.** Show the contents of ATMARP inverse packets exchanged between two routers that have a PVC connection. The IP addresses are 172.14.20.16/16 and 180.25.23.14/24. Choose two arbitrary 20-byte physical addresses. Use hexadecimal values in filling the fields.

**8.** Show the contents of ATMARP packets (request and reply) exchanged between a router and a server. The IP address of the router is 14.56.12.8/16 and the IP address of the server is 200.23.54.8/24. Choose two arbitrary 20-byte physical addresses. Use hexadecimal values in filling the fields.

**9.** Add IP addresses for the routers in Figure 8.12. Note that the prefix in each LIS must be the same, but it must be different for the two LISs. Note also that the routers that belong to two LISs must have two IP addresses.

**10.** An ATMARP packet must also be carried in cells. How many cells are needed to carry an ATMARP packet discussed in this chapter?

CHAPTER

# 9

# *Internet Control Message Protocol Version 4 (ICMPv4)*

As discussed in Chapter 7, the IPv4 provides unreliable and connectionless datagram delivery. It was designed this way to make efficient use of network resources. The IP protocol is a best-effort delivery service that delivers a datagram from its original source to its final destination. However, it has two deficiencies: lack of error control and lack of assistance mechanisms. ICMPv4 is designed to compensate for these deficiencies.

## OBJECTIVES

*The chapter has several objectives:*

- To discuss the rationale for the existence of ICMP.
- To show how ICMP messages are divided into two categories: error-reporting and query messages.
- To discuss the purpose and format of error-reporting messages.
- To discuss the purpose and format of query messages.
- To show how the checksum is calculated for an ICMP message.
- To show how debugging tools using the ICMP protocol.
- To show how a simple software package that implements ICMP is organized.

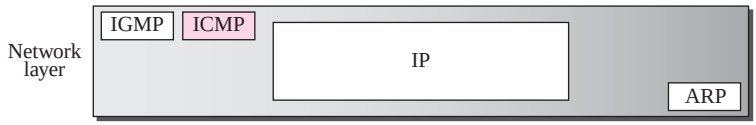
## 9.1 INTRODUCTION

The IP protocol has no error-reporting or error-correcting mechanism. What happens if something goes wrong? What happens if a router must discard a datagram because it cannot find a router to the final destination, or because the time-to-live field has a zero value? What happens if the final destination host must discard all fragments of a datagram because it has not received all fragments within a predetermined time limit? These are examples of situations where an error has occurred and the IP protocol has no built-in mechanism to notify the original host.

The IP protocol also lacks a mechanism for host and management queries. A host sometimes needs to determine if a router or another host is alive. And sometimes a network manager needs information from another host or router.

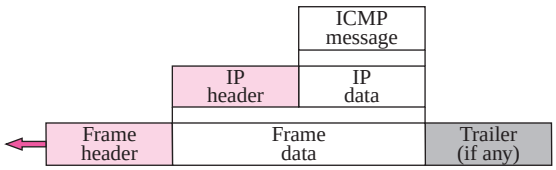
The **Internet Control Message Protocol (ICMP)** has been designed to compensate for the above two deficiencies. It is a companion to the IP protocol. Figure 9.1 shows the position of ICMP in relation to IP and other protocols in the network layer.

**Figure 9.1** *Position of ICMP in the network layer*




ICMP itself is a network layer protocol. However, its messages are not passed directly to the data link layer as would be expected. Instead, the messages are first encapsulated inside IP datagrams before going to the lower layer (see Figure 9.2).

**Figure 9.2** *ICMP encapsulation*




The value of the protocol field in the IP datagram is 1 to indicate that the IP data is an ICMP message.

## 9.2 MESSAGES

ICMP messages are divided into two broad categories: **error-reporting messages** and **query messages.** The error-reporting messages report problems that a router or a host (destination) may encounter when it processes an IP packet. The query messages, which occur in pairs, help a host or a network manager get specific information from a router or another host. For example, nodes can discover their neighbors. Also, hosts can discover and learn about routers on their network and routers can help a node redirect its messages. Table 9.1 lists the ICMP messages in each category.

**Table 9.1** *ICMP messages*

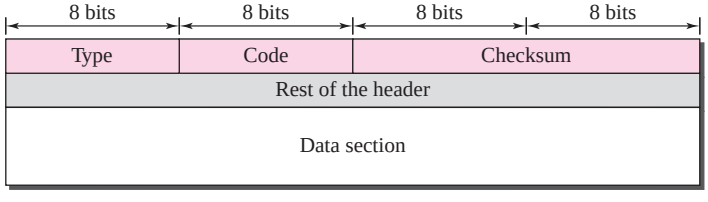
| *Category* | *Type* | *Message* |
|---|---|---|
| Error-reporting messages | 3 | Destination unreachable |
| | 4 | Source quench |
| | 11 | Time exceeded |
| | 12 | Parameter problem |
| | 5 | Redirection |
| Query messages | 8 or 0 | Echo request or reply |
| | 13 or 14 | Timestamp request or reply |

### Message Format

An ICMP message has an 8-byte header and a variable-size data section. Although the general format of the header is different for each message type, the first 4 bytes are common to all. As Figure 9.3 shows, the first field, ICMP type, defines the type of the message. The code field specifies the reason for the particular message type. The last common field is the checksum field (to be discussed later in the chapter). The rest of the header is specific for each message type.

The data section in error messages carries information for finding the original packet that had the error. In query messages, the data section carries extra information based on the type of the query.

**Figure 9.3** *General format of ICMP messages*




### Error Reporting Messages

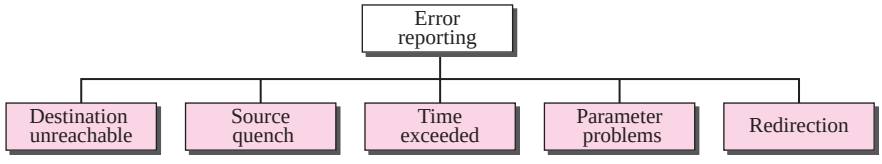
One of the main responsibilities of ICMP is to report errors. Although technology has produced increasingly reliable transmission media, errors still exist and must be handled. IP, as discussed in Chapter 7, is an unreliable protocol. This means that error

checking and error control are not a concern of IP. ICMP was designed, in part, to compensate for this shortcoming. However, ICMP does not correct errors, it simply reports them. Error correction is left to the higher-level protocols. Error messages are always sent to the original source because the only information available in the datagram about the route is the source and destination IP addresses. ICMP uses the source IP address to send the error message to the source (originator) of the datagram.

**ICMP always reports error messages to the original source.**

Five types of errors are handled: destination unreachable, source quench, time exceeded, parameter problems, and redirection (see Figure 9.4).

**Figure 9.4** *Error-reporting messages*




The following are important points about ICMP error messages:

- No ICMP error message will be generated in response to a datagram carrying an ICMP error message.
- No ICMP error message will be generated for a fragmented datagram that is not the first fragment.
- No ICMP error message will be generated for a datagram having a multicast address.
- No ICMP error message will be generated for a datagram having a special address such as 127.0.0.0 or 0.0.0.0.

Note that all error messages contain a data section that includes the IP header of the original datagram plus the first 8 bytes of data in that datagram. The original datagram header is added to give the original source, which receives the error message, information about the datagram itself. The 8 bytes of data are included because, as we will see in Chapters 14 and 15 on UDP and TCP protocols, the first 8 bytes provide information about the port numbers (UDP and TCP) and sequence number (TCP). This information is needed so the source can inform the protocols (TCP or UDP) about the error. ICMP forms an error packet, which is then encapsulated in an IP datagram (see Figure 9.5).

### *Destination Unreachable*

When a router cannot route a datagram or a host cannot deliver a datagram, the datagram is discarded and the router or the host sends a **destination-unreachable message** back to the source host that initiated the datagram. Figure 9.6 shows the format of the

**Figure 9.5** *Contents of data field for the error messages*

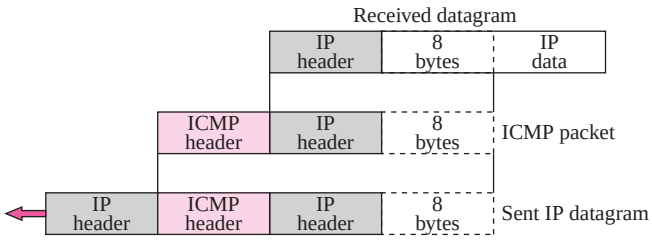
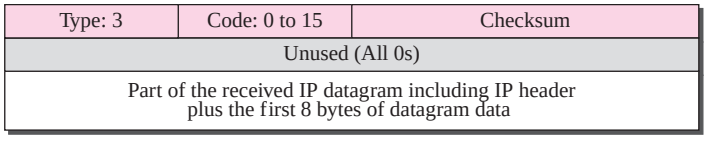



**Figure 9.6** *Destination-unreachable format*




destination-unreachable message. The code field for this type specifies the reason for discarding the datagram:

- **Code 0.** The network is unreachable, possibly due to hardware failure.
- **Code 1.** The host is unreachable. This can also be due to hardware failure.
- **Code 2.** The protocol is unreachable. An IP datagram can carry data belonging to higher-level protocols such as UDP, TCP, and OSPF. If the destination host receives a datagram that must be delivered, for example, to the TCP protocol, but the TCP protocol is not running at the moment, a code 2 message is sent.
- **Code 3.** The port is unreachable. The application program (process) that the datagram is destined for is not running at the moment.
- **Code 4.** Fragmentation is required, but the DF (do not fragment) field of the datagram has been set. In other words, the sender of the datagram has specified that the datagram not be fragmented, but routing is impossible without fragmentation.
- **Code 5.** Source routing cannot be accomplished. In other words, one or more routers defined in the source routing option cannot be visited.
- **Code 6.** The destination network is unknown. This is different from code 0. In code 0, the router knows that the destination network exists, but it is unreachable at the moment. For code 6, the router has no information about the destination network.
- **Code 7.** The destination host is unknown. This is different from code 1. In code 1, the router knows that the destination host exists, but it is unreachable at the moment. For code 7, the router is unaware of the existence of the destination host.

- **Code 8.** The source host is isolated.
- **Code 9.** Communication with the destination network is administratively prohibited.
- **Code 10.** Communication with the destination host is administratively prohibited.
- **Code 11.** The network is unreachable for the specified type of service. This is different from code 0. Here the router can route the datagram if the source had requested an available type of service.
- **Code 12.** The host is unreachable for the specified type of service. This is different from code 1. Here the router can route the datagram if the source had requested an available type of service.
- **Code 13.** The host is unreachable because the administrator has put a filter on it.
- **Code 14.** The host is unreachable because the host precedence is violated. The message is sent by a router to indicate that the requested precedence is not permitted for the destination.
- **Code 15.** The host is unreachable because its precedence was cut off. This message is generated when the network operators have imposed a minimum level of precedence for the operation of the network, but the datagram was sent with a precedence below this level.

Note that destination-unreachable messages can be created either by a router or the destination host. Code 2 and code 3 messages can only be created by the destination host; the messages of the remaining codes can only be created by routers.

**Destination-unreachable messages with codes 2 or 3 can be created only by the destination host. Other destination-unreachable messages can be created only by routers.**

Note that even if a router does not report a destination-unreachable message, it does not necessarily mean that the datagram has been delivered. For example, if a datagram is traveling through an Ethernet network, there is no way that a router knows that the datagram has been delivered to the destination host or the next router because Ethernet does not provide any acknowledgment mechanism.

**A router cannot detect all problems that prevent the delivery of a packet.**

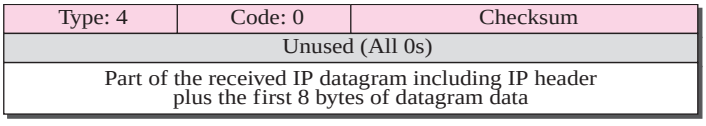
### *Source Quench*

The IP protocol is a connectionless protocol. There is no communication between the source host, which produces the datagram, the routers, which forward it, and the destination host, which processes it. One of the ramifications of this absence of communication is the lack of *flow control* and *congestion control*.

**There is no flow-control or congestion-control mechanism in the IP protocol.**

The **source-quench message** in ICMP was designed to add a kind of flow control and congestion control to the IP. When a router or host discards a datagram due to congestion, it sends a source-quench message to the sender of the datagram. This message has two purposes. First, it informs the source that the datagram has been discarded. Second, it warns the source that there is congestion somewhere in the path and that the source should slow down (quench) the sending process. The source-quench format is shown in Figure 9.7.

**Figure 9.7** *Source-quench format*




**A source-quench message informs the source that a datagram has been discarded due to congestion in a router or the destination host. The source must slow down the sending of datagrams until the congestion is relieved.**

There are some points that deserve more explanation. First, the router or destination host that has experienced the congestion sends one source-quench message for each discarded datagram to the source host. Second, there is no mechanism to tell the source that the congestion has been relieved and the source can resume sending datagrams at its previous rate. The source continues to lower the rate until no more source-quench messages are received. Third, the congestion can be created either by a one-to-one or many-to-one communication. In a one-to-one communication, a single high-speed host could create datagrams faster than a router or the destination host can handle. In this case, source-quench messages can be helpful. They tell the source to slow down. In a many-to-one communication, many sources create datagrams that must be handled by a router or the destination host. In this case, each source can be sending datagrams at different speeds, some of them at a low rate, others at a high rate. Here, the source-quench message may not be very useful. The router or the destination host has no clue which source is responsible for the congestion. It may drop a datagram from a very slow source instead of dropping the datagram from the source that has actually created the congestion.

**One source-quench message is sent for each datagram that is discarded due to congestion.**

### *Time Exceeded*

The **time-exceeded message** is generated in two cases:

- First, as we saw in Chapter 6, routers use routing tables to find the next hop (next router) that must receive the packet. If there are errors in one or more routing tables, a packet can travel in a loop or a cycle, going from one router to the next or visiting a series of routers endlessly. As we saw in Chapter 7, each datagram contains a field called *time to live* that controls this situation. When a datagram visits a router, the value of this field is decremented by 1. When the time-to-live value reaches 0, after decrementing, the router discards the datagram. However, when the datagram is discarded, a time-exceeded message must be sent by the router to the original source.

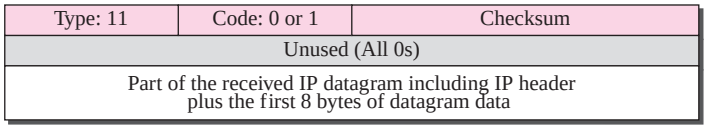
**Whenever a router decrements a datagram with a time-to-live value to zero, it discards the datagram and sends a time-exceeded message to the original source.**

- Second, a time-exceeded message is also generated when all fragments that make up a message do not arrive at the destination host within a certain time limit. When the first fragment arrives, the destination host starts a timer. If all the fragments have not arrived when the time expires, the destination discards all the fragments and sends a time-exceeded message to the original sender.

**When the final destination does not receive all of the fragments in a set time, it discards the received fragments and sends a time-exceeded message to the original source.**

Figure 9.8 shows the format of the time-exceeded message. Code 0 is used when the datagram is discarded by the router due to a time-to-live field value of zero. Code 1 is used when arrived fragments of a datagram are discarded because some fragments have not arrived within the time limit.

**Figure 9.8** *Time-exceeded message format*




**In a time-exceeded message, code 0 is used only by routers to show that the value of the time-to-live field is zero. Code 1 is used only by the destination host to show that not all of the fragments have arrived within a set time.**

### Parameter Problem

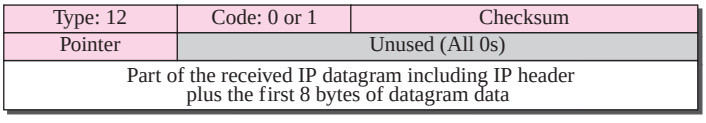
Any ambiguity in the header part of a datagram can create serious problems as the datagram travels through the Internet. If a router or the destination host discovers an ambiguous or missing value in any field of the datagram, it discards the datagram and sends a parameter-problem message back to the source.

**A parameter-problem message can be created by a router or the destination host.**

Figure 9.9 shows the format of the **parameter-problem message.** The code field in this case specifies the reason for discarding the datagram:

- **Code 0.** There is an error or ambiguity in one of the header fields. In this case, the value in the pointer field points to the byte with the problem. For example, if the value is zero, then the first byte is not a valid field.
- **Code 1.** The required part of an option is missing. In this case, the pointer is not used.

**Figure 9.9** *Parameter-problem message format*

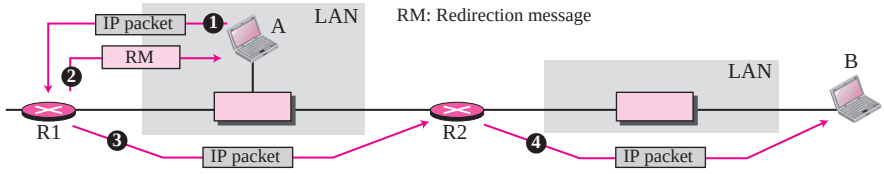



### Redirection

When a router needs to send a packet destined for another network, it must know the IP address of the next appropriate router. The same is true if the sender is a host. Both routers and hosts then must have a routing table to find the address of the router or the next router. Routers take part in the routing update process as we will see in Chapter 11 and are supposed to be updated constantly. Routing is dynamic.

However, for efficiency, hosts do not take part in the routing update process because there are many more hosts in an internet than routers. Updating the routing tables of hosts dynamically produces unacceptable traffic. The hosts usually use static routing. When a host comes up, its routing table has a limited number of entries. It usually knows only the IP address of one router, the default router. For this reason, the host may send a datagram, which is destined for another network, to the wrong router. In this case, the router that receives the datagram will forward the datagram to the correct router. However, to update the routing table of the host, it sends a redirection message to the host. This concept of redirection is shown in Figure 9.10. Host A wants to send a datagram to host B. Router R2 is obviously the most efficient routing choice, but host A did not choose router R2. The datagram goes to R1 instead. R1, after consulting its table, finds that the packet should have gone to R2. It sends the packet to R2 and, at the same time, sends a redirection message to host A. Host A's routing table can now be updated.

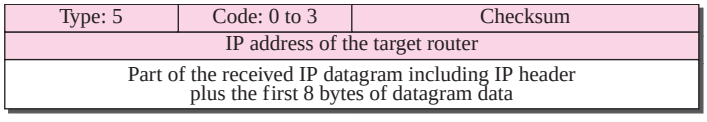
**Figure 9.10** *Redirection concept*




**A host usually starts with a small routing table that is gradually augmented and updated. One of the tools to accomplish this is the redirection message.**

The format of the **redirection message** is shown in Figure 9.11. Note that the IP address of the appropriate target is given in the second row.

**Figure 9.11** *Redirection message format*




Although the redirection message is considered an error-reporting message, it is different from other error messages. The router does not discard the datagram in this case; it is sent to the appropriate router. The code field for the redirection message narrows down the redirection:

- **Code 0.** Redirection for a network-specific route.
- **Code 1.** Redirection for a host-specific route.
- **Code 2.** Redirection for a network-specific route based on a specified type of service.
- **Code 3.** Redirection for a host-specific route based on a specified type of service.

**A redirection message is sent from a router to a host on the same local network.**

## Query Messages

In addition to error reporting, ICMP can also diagnose some network problems. This is accomplished through the query messages. A group of five different pairs of messages have been designed for this purpose, but three of these pairs are deprecated today, as we discuss later in the section. Only two pairs are used today: echo request and replay and timestamp request and replay. In this type of ICMP message, a node sends a message that is answered in a specific format by the destination node.

### *Echo Request and Reply*

The **echo-request** and **echo-reply messages** are designed for diagnostic purposes. Network managers and users utilize this pair of messages to identify network problems. The combination of echo-request and echo-reply messages determines whether two systems (hosts or routers) can communicate with each other.

A host or router can send an echo-request message to another host or router. The host or router that receives an echo-request message creates an echo-reply message and returns it to the original sender.

**An echo-request message can be sent by a host or router. An echo-reply message is sent by the host or router that receives an echo-request message.**

The echo-request and echo-reply messages can be used to determine if there is communication at the IP level. Because ICMP messages are encapsulated in IP datagrams, the receipt of an echo-reply message by the machine that sent the echo request is proof that the IP protocols in the sender and receiver are communicating with each other using the IP datagram. Also, it is proof that the intermediate routers are receiving, processing, and forwarding IP datagrams.

**Echo-request and echo-reply messages can be used by network managers to check the operation of the IP protocol.**

The echo-request and echo-reply messages can also be used by a host to see if another host is reachable. At the user level, this is done by invoking the packet Internet groper (ping) command. Today, most systems provide a version of the *ping* command that can create a series (instead of just one) of echo-request and echo-reply messages, providing statistical information. We will see the use of this program at the end of the chapter.

**Echo-request and echo-reply messages can test the reachability of a host. This is usually done by invoking the ping command.**

Echo request, together with echo reply, can determine whether or not a node is functioning properly. The node to be tested is sent an echo-request message. The optional data field contains a message that must be repeated exactly by the responding node in its echo-reply message. Figure 9.12 shows the format of the echo-reply and echo-request message. The identifier and sequence number fields are not formally defined by the protocol and can be used arbitrarily by the sender. The identifier is often the same as the process ID.

### *Timestamp Request and Reply*

Two machines (hosts or routers) can use the **timestamp-request** and **timestamp-reply messages** to determine the round-trip time needed for an IP datagram to travel between

**Figure 9.12** *Echo-request and echo-reply messages*

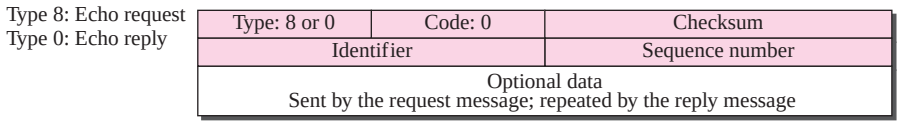
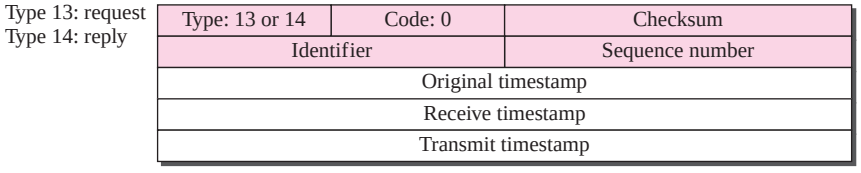



**Figure 9.13** *Timestamp-request and timestamp-reply message format*




them. It can also be used to synchronize the clocks in two machines. The format of these two messages is shown in Figure 9.13.

The three timestamp fields are each 32 bits long. Each field can hold a number representing time measured in milliseconds from midnight in Universal Time (formerly called Greenwich Mean Time). (Note that 32 bits can represent a number between 0 and 4,294,967,295, but a timestamp in this case cannot exceed $86,400,000 = 24 \times 60 \times 60 \times 1000$.)

The source creates a timestamp-request message. The source fills the *original timestamp* field with the Universal Time shown by its clock at departure time. The other two timestamp fields are filled with zeros.

The destination creates the timestamp-reply message. The destination copies the original timestamp value from the request message into the same field in its reply message. It then fills the *receive timestamp* field with the Universal Time shown by its clock at the time the request was received. Finally, it fills the *transmit timestamp* field with the Universal Time shown by its clock at the time the reply message departs.

The timestamp-request and timestamp-reply messages can be used to compute the one-way or round-trip time required for a datagram to go from a source to a destination and then back again. The formulas are

**sending time = receive timestamp – original timestamp**
**receiving time = returned time – transmit timestamp**
**round-trip time = sending time + receiving time**

Note that the sending and receiving time calculations are accurate only if the two clocks in the source and destination machines are synchronized. However, the round-trip calculation is correct even if the two clocks are not synchronized because each clock contributes twice to the round-trip calculation, thus canceling any difference in synchronization.

**Timestamp-request and timestamp-reply messages can be used to calculate the round-trip time between a source and a destination machine even if their clocks are not synchronized.**

For example, given the following information:

| | |
|---|---|
| **original timestamp: 46** | **receive timestamp: 59** |
| **transmit timestamp: 60** | **return time: 67** |

We can calculate the round-trip time to be 20 milliseconds:

**sending time = 59 − 46 = 13 milliseconds**
**receiving time = 67 − 60 = 7 milliseconds**
**round-trip time = 13 + 7 = 20 milliseconds**

Given the actual one-way time, the timestamp-request and timestamp-reply messages can also be used to synchronize the clocks in two machines using the following formula:

**Time difference = receive timestamp − (original timestamp field + one-way time duration)**

The one-way time duration can be obtained either by dividing the round-trip time duration by two (if we are sure that the sending time is the same as the receiving time) or by other means. For example, we can tell that the two clocks in the previous example are 3 milliseconds out of synchronization because

**Time difference = 59 − (46 + 10) = 3**

**The timestamp-request and timestamp-reply messages can be used to synchronize two clocks in two machines if the exact one-way time duration is known.**

### *Deprecated Messages*

Three pairs of messages are declared obsolete by IETF:

1. *Information request and replay* messages are not used today because their duties are done by Address Resolution Protocol (ARP) discussed in Chapter 8.
2. *Address mask request and reply* messages are not used today because their duties are done by Dynamic Host Configuration Protocol (DHCP), discussed in Chapter 8.
3. *Router solicitation and advertisment* messages are not used today because their duties are done by Dynamic Host Configuration Protocol (DHCP), discussed in Chapter 18.

## Checksum

In Chapter 7, we learned the concept and idea of the checksum. In ICMP the checksum is calculated over the entire message (header and data).

### *Checksum Calculation*

The sender follows these steps using one's complement arithmetic:

1. The checksum field is set to zero.
2. The sum of all the 16-bit words (header and data) is calculated.
3. The sum is complemented to get the checksum.
4. The checksum is stored in the checksum field.

### *Checksum Testing*

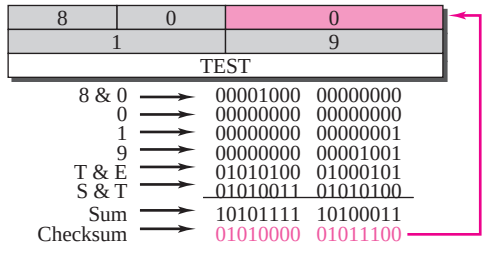
The receiver follows these steps using one's complement arithmetic:

1. The sum of all words (header and data) is calculated.
2. The sum is complemented.
3. If the result obtained in step 2 is 16 0s, the message is accepted; otherwise, it is rejected.

### Example 9.1

Figure 9.14 shows an example of checksum calculation for a simple echo-request message (see Figure 9.12). We randomly chose the identifier to be 1 and the sequence number to be 9. The message is divided into 16-bit (2-byte) words. The words are added together and the sum is complemented. Now the sender can put this value in the checksum field.

**Figure 9.14** *Example of checksum calculation*




## 9.3 DEBUGGING TOOLS

There are several tools that can be used in the Internet for debugging. We can find if a host or router is alive and running. We can trace the route of a packet. We introduce two tools that use ICMP for debugging: *ping* and *traceroute*.We will introduce more tools in future chapters after we have discussed the corresponding protocols.

### Ping

We can use the **ping** program to find if a host is alive and responding. We used the *ping* program in Chapter 7 to simulate the record route option. We discuss *ping* in more detail to see how it uses ICMP packets.

The source host sends ICMP echo request messages (type: 8, code: 0); the destination, if alive, responds with ICMP echo reply messages. The *ping* program sets the identifier field in the echo request and reply message and starts the sequence number from 0; this number is incremented by one each time a new message is sent. Note that *ping* can calculate the round-trip time. It inserts the sending time in the data section of the message. When the packet arrives it subtracts the arrival time from the departure time to get the **round-trip time** (RTT).

### Example 9.2

We use the *ping* program to test the server fhda.edu. The result is shown below:

```
$ ping fhda.edu
PING fhda.edu (153.18.8.1)  56 (84) bytes of data.
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=0   ttl=62   time=1.91 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=1   ttl=62   time=2.04 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=2   ttl=62   time=1.90 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=3   ttl=62   time=1.97 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=4   ttl=62   time=1.93 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=5   ttl=62   time=2.00 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=6   ttl=62   time=1.94 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=7   ttl=62   time=1.94 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=8   ttl=62   time=1.97 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=9   ttl=62   time=1.89 ms
64 bytes from tiptoe.fhda.edu (153.18.8.1): icmp_seq=10  ttl=62   time=1.98 ms

--- fhda.edu ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10103 ms
rtt min/avg/max = 1.899/1.955/2.041 ms
```

The *ping* program sends messages with sequence numbers starting from 0. For each probe it gives us the RTT time. The TTL (time to live) field in the IP datagram that encapsulates an ICMP message has been set to 62, which means the packet cannot travel more than 62 hops. At the beginning, *ping* defines the number of data bytes as 56 and the total number of bytes as 84. It is obvious that if we add 8 bytes of ICMP header and 20 bytes of IP header to 56, the result is 84. However, note that in each probe *ping* defines the number of bytes as 64. This is the total number of bytes in the ICMP packet (56 + 8). The *ping* program continues to send messages if we do not stop it using the interrupt key (ctrl + c, for example). After it is interrupted, it prints the statistics of the probes. It tells us the number of packets sent, the number of packets received, the total time, and the RTT minimum, maximum, and average. Some systems may print more information.

### Example 9.3

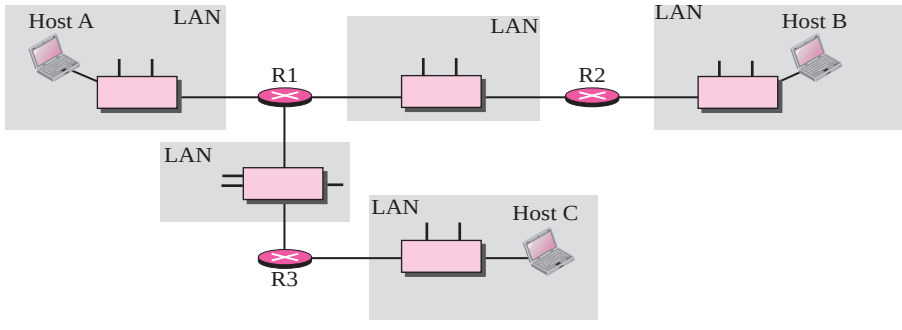
For the second example, we want to know if the adelphia.net mail server is alive and running. The result is shown below: Note that in this case, we sent 14 packets, but only 13 have been returned. We may have interrupted the program before the last packet, with sequence number 13, was returned.

```
$ ping mail.adelphia.net
PING mail.adelphia.net (68.168.78.100) 56(84) bytes of data.
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=0   ttl=48   time=85.4 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=1   ttl=48   time=84.6 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=2   ttl=48   time=84.9 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=3   ttl=48   time=84.3 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=4   ttl=48   time=84.5 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=5   ttl=48   time=84.7 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=6   ttl=48   time=84.6 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=7   ttl=48   time=84.7 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=8   ttl=48   time=84.4 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=9   ttl=48   time=84.2 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=10  ttl=48   time=84.9 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=11  ttl=48   time=84.6 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=12  ttl=48   time=84.5 ms

--- mail.adelphia.net ping statistics ---
14 packets transmitted, 13 received, 7% packet loss, time 13129 ms
rtt min/avg/max/mdev = 84.207/84.694/85.469
```

## Traceroute

The ***traceroute*** program in UNIX or ***tracert*** in Windows can be used to trace the route of a packet from the source to the destination. Let us show the idea of the **traceroute** program using Figure 9.15.

**Figure 9.15** *The traceroute program operation*




We have seen an application of the *traceroute* program to simulate the loose source route and strict source route options of an IP datagram in the previous chapter. We use this program in conjunction with ICMP packets in this chapter. The program elegantly uses two ICMP messages, time exceeded and destination unreachable, to find the route of a packet. This is a program at the application level that uses the services of UDP (see Chapter 14).

Given the topology, we know that a packet from host A to host B travels through routers R1 and R2. However, most of the time, we are not aware of this topology. There

could be several routes from A to B. The *traceroute* program uses the ICMP messages and the TTL (time to live) field in the IP packet to find the route.

1. The traceroute program uses the following steps to find the address of the router R1 and the round trip time between host A and router R1. The *traceroute* program repeats steps a to c three times to get a better average round-trip time. The first trip time may be much longer than the second or third because it takes time for the ARP program to find the physical address of router R1. For the second and third trip, ARP has the address in its cache.
   a. The traceroute application at host A sends a packet to destination B using UDP; the message is encapsulated in an IP packet with a TTL value of 1. The program notes the time the packet is sent.
   b. Router R1 receives the packet and decrements the value of TTL to 0. It then discards the packet (because TTL is 0). The router, however, sends a time-exceeded ICMP message (type: 11, code: 0) to show that the TTL value is 0 and the packet was discarded.
   c. The traceroute program receives the ICMP messages and uses the source address of the IP packet encapsulating ICMP to find the IP address of router R1. The program also makes note of the time the packet has arrived. The difference between this time and the time at step a is the round-trip time.
2. The traceroute program repeats the previous steps to find the address of router R2 and the round-trip time between host A and router R2. However, in this step, the value of TTL is set to 2. So router R1 forwards the message, while router R2 discards it and sends a time-exceeded ICMP message.
3. The traceroute program repeats the previous step to find the address of host B and the round-trip time between host A and host B. When host B receives the packet, it decrements the value of TTL, but it does not discard the message since it has reached its final destination. How can an ICMP message be sent back to host A? The traceroute program uses a different strategy here. The destination port of the UDP packet is set to one that is not supported by the UDP protocol. When host B receives the packet, it cannot find an application program to accept the delivery. It discards the packet and sends an ICMP destination-unreachable message (type: 3, code: 3) to host A. Note that this situation does not happen at router R1 or R2 because a router does not check the UDP header. The traceroute program records the destination address of the arrived IP datagram and makes note of the round-trip time. Receiving the destination-unreachable message with a code value 3 is an indication that the whole route has been found and there is no need to send more packets.

### Example 9.4

We use the traceroute program to find the route from the computer voyager.deanza.edu to the server fhda.edu. The following shows the result.

```
$ traceroute fhda.edu
traceroute to fhda.edu    (153.18.8.1), 30 hops max, 38 byte packets
1 Dcore.fhda.edu          (153.18.31.25)    0.995 ms    0.899 ms    0.878 ms
2 Dbackup.fhda.edu        (153.18.251.4)    1.039 ms    1.064 ms    1.083 ms
3  tiptoe.fhda.edu        (153.18.8.1)      1.797 ms    1.642 ms    1.757 ms
```

The unnumbered line after the command shows that the destination is 153.18.8.1. The TTL value is 30 hops. The packet contains 38 bytes: 20 bytes of IP header, 8 bytes of UDP header, and 10 bytes of application data. The application data is used by traceroute to keep track of the packets. The first line shows the first router visited. The router is named Dcore.fhda.edu with IP address 153.18.31.254. The first round-trip time was 0.995 milliseconds, the second was 0.899 milliseconds, and the third was 0.878 milliseconds. The second line shows the second router visited. The router is named Dbackup.fhda.edu with IP address 153.18.251.4. The three round-trip times are also shown. The third line shows the destination host. We know that this is the destination host because there are no more lines. The destination host is the server fhda.edu, but it is named tiptoe.fhda.edu with the IP address 153.18.8.1. The three round-trip times are also shown.

### Example 9.5

In this example, we trace a longer route, the route to xerox.com

```
$ traceroute xerox.com
traceroute to xerox.com (13.1.64.93), 30 hops max, 38 byte packets
1  Dcore.fhda.edu      (153.18.31.254)     0.622 ms    0.891 ms    0.875 ms
2  Ddmz.fhda.edu       (153.18.251.40)     2.132 ms    2.266 ms    2.094 ms
3  Cinic.fhda.edu      (153.18.253.126)    2.110 ms    2.145 ms    1.763 ms
4  cenic.net           (137.164.32.140)    3.069 ms    2.875 ms    2.930 ms
5  cenic.net           (137.164.22.31)     4.205 ms    4.870 ms    4.197 ms
6  cenic.net           (137.164.22.167)    4.250 ms    4.159 ms    4.078 ms
7  cogentco.com        (38.112.6.225)      5.062 ms    4.825 ms    5.020 ms
8  cogentco.com        (66.28.4.69)        6.070 ms    6.207 ms    5.653 ms
9  cogentco.com        (66.28.4.94)        6.070 ms    5.928 ms    5.499 ms
10 cogentco.com        (154.54.2.226)      6.545 ms    6.399 ms    6.535 ms
11 sbcglo              (151.164.89.241)    6.379 ms    6.370 ms    6.379 ms
12 sbcglo              (64.161.1.45)       6.908 ms    6.748 ms    7.359 ms
13 sbcglo              (64.161.1.29)       7.023 ms    7.040 ms    6.734 ms
14 snfc21.pbi.net      (151.164.191.49)    7.656 ms    7.129 ms    6.866 ms
15 sbcglobal.net       (151.164.243.58)    7.844 ms    7.545 ms    7.353 ms
16 pacbell.net         (209.232.138.114)   9.857 ms    9.535 ms    9.603 ms
17 209.233.48.223      (209.233.48.223)    10.634 ms   10.771 ms   10.592 ms
18 alpha.Xerox.COM     (13.1.64.93)        10.922 ms   11.048 ms   10.922 ms
```

Here there are 17 hops between source and destination. Note that some round-trip times look unusual. It could be that a router is too busy to process the packet immediately.

### Example 9.6

An interesting point is that a host can send a traceroute packet to itself. This can be done by specifying the host as the destination. The packet goes to the loopback address as we expect.

```
$ traceroute voyager.deanza.edu
traceroute to voyager.deanza.edu  (127.0.0.1), 30 hops max, 38 byte packets
1  voyager        (127.0.0.1)          0.178 ms          0.086 ms          0.055 ms
```

### Example 9.7

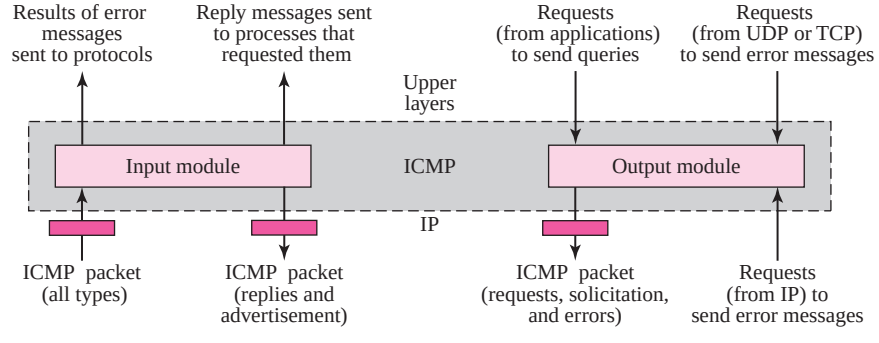
Finally, we use the traceroute program to find the route between fhda.edu and mhhe.com (McGraw-Hill server). We notice that we cannot find the whole route. When traceroute does not receive a response within 5 seconds, it prints an asterisk to signify a problem (not the case in this example), and then tries the next hop.

```
$ traceroute mhhe.com
traceroute to mhhe.com (198.45.24.104), 30 hops max, 38 byte packets
1   Dcore.fhda.edu      (153.18.31.254)      1.025 ms     0.892 ms     0.880 ms
2   Ddmz.fhda.edu       (153.18.251.40)      2.141 ms     2.159 ms     2.103 ms
3   Cinic.fhda.edu      (153.18.253.126)     2.159 ms     2.050 ms     1.992 ms
4   cenic.net           (137.164.32.140)     3.220 ms     2.929 ms     2.943 ms
5   cenic.net           (137.164.22.59)      3.217 ms     2.998 ms     2.755 ms
6   SanJose1.net        (209.247.159.109)    10.653 ms    10.639 ms    10.618 ms
7   SanJose2.net        (64.159.2.1)         10.804 ms    10.798 ms    10.634 ms
8   Denver1.Level3.net  (64.159.1.114)       43.404 ms    43.367 ms    43.414 ms
9   Denver2.Level3.net  (4.68.112.162)       43.533 ms    43.290 ms    43.347 ms
10  unknown             (64.156.40.134)      55.509 ms    55.462 ms    55.647 ms
11  mcleodusa1.net      (64.198.100.2)       60.961 ms    55.681 ms    55.461 ms
12  mcleodusa2.net      (64.198.101.202)     55.692 ms    55.617 ms    55.505 ms
13  mcleodusa3.net      (64.198.101.142)     56.059 ms    55.623 ms    56.333 ms
14  mcleodusa4.net      (209.253.101.178)    297.199 ms   192.790 ms   250.594 ms
15  eppg.com            (198.45.24.246)      71.213 ms    70.536 ms    70.663 ms
16  …                   …                    …            …            …
```

## 9.4 ICMP PACKAGE

To give an idea of how ICMP can handle the sending and receiving of ICMP messages, we present our version of an ICMP package made of two modules: an input module and an output module. Figure 9.16 shows these two modules.

**Figure 9.16** *ICMP package*



## Input Module

The input module handles all received ICMP messages. It is invoked when an ICMP packet is delivered to it from the IP layer. If the received packet is a request, the module creates a reply and sends it out. If the received packet is a redirection message, the module uses the information to update the routing table. If the received packet is an error message, the module informs the protocol about the situation that caused the error. The pseudocode is shown below:

**Table 9.2** *Input Module*

```
ICMP_Input_module (ICMP_Packet)
{
   If (the type is a request)
   {
        Create a reply
        Send the reply
   }
   If (the type defines a redirection)
   {
       Modify the routing table
   }
   If (the type defines other error messages)
   {
       Inform the appropriate source protocol
   }
   Return
}
```

## Output Module

The output module is responsible for creating request, solicitation, or error messages requested by a higher level or the IP protocol. The module receives a demand from IP, UDP, or TCP to send one of the ICMP error messages. If the demand is from IP, the output module must first check that the request is allowed. Remember, an ICMP message cannot be created for four situations: an IP packet carrying an ICMP error message, a fragmented IP packet, a multicast IP packet, or an IP packet having IP address 0.0.0.0 or 127.X.Y. Z. The output module may also receive a demand from an application program to send one of the ICMP request messages. The pseudocode is shown in Table 9.3.

**Table 9.3** *Output Module*

```
ICMP_Output_Module (demand)
{
   If (the demand defines an error message)
   {
```

**Table 9.3** *Output Module (continued)*

```
5              If (demand comes from IP AND is forbidden)
6              {
7                   Return
8              }
9              If (demand is a valid redirection message)
10             {
11                  Return
12             }
13             Create an error message
14        If (demand defines a request)
15        {
16             Create a request message
17        }
18        Send the message
19        Return
20   }
```

## 9.5 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], and [Ste 94].

### RFCs

Several RFCs discuss ICMP including RFC 792, RFC 950, RFC 956, RFC 957, RFC 1016, RFC 1122, RFC 1256, RFC 1305, and RFC 1987.

## 9.6 KEY TERMS

destination-unreachable message
echo-reply messages
error-reporting message
echo-request message
Internet Control Message Protocol (ICMP)
parameter-problem message
ping
query message
redirection message
round-trip time (RTT)
source-quench message
time-exceeded message
timestamp-reply message
timestamp-request message
traceroute

## 9.7 SUMMARY

- The Internet Control Message Protocol (ICMP) supports the unreliable and connectionless Internet Protocol (IP).
- ICMP messages are encapsulated in IP datagrams. There are two categories of ICMP messages: error-reporting and query messages. The error-reporting messages report problems that a router or a host (destination) may encounter when it processes an IP packet. The query messages, which occur in pairs, help a host or a network manager get specific information from a router or another host.
- The checksum for ICMP is calculated using both the header and the data fields of the ICMP message.
- There are several tools that can be used in the Internet for debugging. We can find if a host or router is alive and running. Two of these tools are *ping* and *traceroute*.
- A simple ICMP design can consist of an input module that handles incoming ICMP packets and an output module that handles demands for ICMP services.

## 9.8 PRACTICE SET

### Exercises

1. Host A sends a timestamp-request message to host B and never receives a reply. Discuss three possible causes and the corresponding course of action.
2. Why is there a restriction on the generation of an ICMP message in response to a failed ICMP error message?
3. Host A sends a datagram to host B. Host B never receives the datagram and host A never receives notification of failure. Give two different explanations of what might have happened.
4. What is the purpose of including the IP header and the first 8 bytes of datagram data in the error reporting ICMP messages?
5. What is the maximum value of the pointer field in a parameter-problem message?
6. Give an example of a situation in which a host would never receive a redirection message.
7. Make a table showing which ICMP messages are sent by routers, which are sent by the nondestination hosts, and which are sent by the destination hosts.
8. Can the calculated sending time, receiving time, or round-trip time have a negative value? Why or why not? Give examples.
9. Why isn't the one-way time for a packet simply the round-trip time divided by two?
10. What is the minimum size of an ICMP packet? What is the maximum size of an ICMP packet?

**11.** What is the minimum size of an IP packet that carries an ICMP packet? What is the maximum size?

**12.** What is the minimum size of an Ethernet frame that carries an IP packet which in turn carries an ICMP packet? What is the maximum size?

**13.** How can we determine if an IP packet is carrying an ICMP packet?

**14.** Calculate the checksum for the following ICMP packet:
Type: Echo Request Identifier: 123 Sequence Number: 25 Message: Hello

**15.** A router receives an IP packet with source IP address 130.45.3.3 and destination IP address 201.23.4.6. The router cannot find the destination IP address in its routing table. Fill in the fields (as much as you can) for the ICMP message sent.

**16.** TCP receives a segment with destination port address 234. TCP checks and cannot find an open port for this destination. Fill in the fields for the ICMP message sent.

**17.** An ICMP message has arrived with the header (in hexadecimal):

**03 0310 20 00 00 00 00**

What is the type of the message? What is the code? What is the purpose of the message?

**18.** An ICMP message has arrived with the header (in hexadecimal):

**05 00 11 12 11 0B 03 02**

What is the type of the message? What is the code? What is the purpose of the message? What is the value of the last 4 bytes? What do the last bytes signify?

**19.** A computer sends a timestamp request. If its clock shows 5:20:30 A.M. (Universal Time), show the entries for the message.

**20.** Repeat Exercise 19 for the time of 3:40:30 P.M. (Universal Time).

**21.** A computer receives a timestamp request from another computer at 2:34:20 P.M. The value of the original timestamp is 52,453,000. If the sender clock is 5 ms slow, what is the one-way time?

**22.** A computer sends a timestamp request to another computer. It receives the corresponding timestamp reply at 3:46:07 A.M. The values of the original timestamp, receive timestamp, and transmit timestamp are 13,560,000, 13,562,000, and 13,564,300, respectively. What is the sending trip time? What is the receiving trip time? What is the round-trip time? What is the difference between the sender clock and the receiver clock?

**23.** If two computers are 5000 miles apart, what is the minimum time for a message to go from one to the other?

## Research Activities

**24.** Use the ping program to test your own computer (loopback).

**25.** Use the ping program to test a host inside the United States.

**26.** Use the ping program to test a host outside the United States.

**27.** Use traceroute (or tracert) to find the route from your computer to a computer in a college or university.

**28.** Show how you can find the RTT between two routers using Exercise 27.

CHAPTER

# 10 Mobile IP

Mobile communication has received a lot of attention in the last decade. The interest in mobile communication on the Internet means that the IP protocol, originally designed for stationary devices, must be enhanced to allow the use of mobile computers, computers that move from one network to another.

## OBJECTIVES

*The chapter has several objectives:*

- To discuss addressing issues related to a mobile host and the need for a care-of address.
- To discuss two agents involved in mobile IP communication, the home agent and the foreign agent, and how they communicate.
- To explain three phases of communication between a mobile host and a remote host: agent discovery, registration, and data transfer.
- To mention inefficiency of mobile IP in two cases, double crossing and triangular routing, and a possible solution.

## 10.1 ADDRESSING

The main problem that must be solved in providing mobile communication using the IP protocol is addressing.

### Stationary Hosts

The original IP addressing was based on the assumption that a host is stationary, attached to one specific network. A router uses an IP address to route an IP datagram. As we learned in Chapter 5, an IP address has two parts: a prefix and a suffix. The prefix associates a host to a network. For example, the IP address 10.3.4.24/8 defines a host attached to the network 10.0.0.0/8. This implies that a host in the Internet does not have an address that it can carry with itself from one place to another. The address is valid only when the host is attached to the network. If the network changes, the address is no longer valid. Routers use this association to route a packet; they use the prefix to deliver the packet to the network to which the host is attached. This scheme works perfectly with **stationary hosts.**

**The IP addresses are designed to work with stationary hosts because part of the address defines the network to which the host is attached.**

### Mobile Hosts

When a host moves from one network to another, the IP addressing structure needs to be modified. Several solutions have been proposed.

#### *Changing the Address*

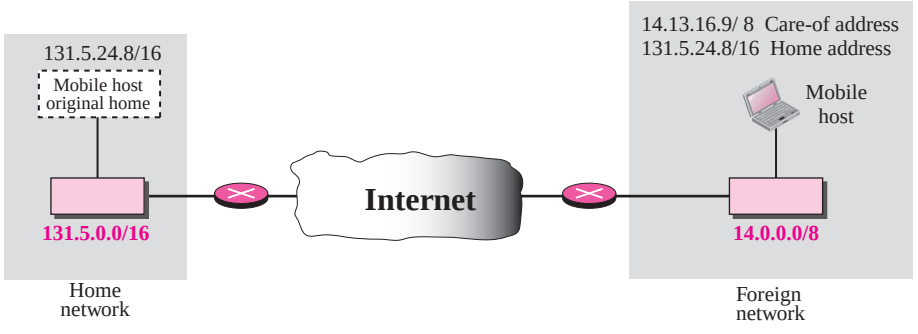
One simple solution is to let the **mobile host** change its address as it goes to the new network. The host can use DHCP (see Chapter 18) to obtain a new address to associate it with the new network. This approach has several drawbacks. First, the configuration files would need to be changed. Second, each time the computer moves from one network to another, it must be rebooted. Third, the DNS tables (see Chapter 19) need to be revised so that every other host in the Internet is aware of the change. Fourth, if the host roams from one network to another during a transmission, the data exchange will be interrupted. This is because the ports and IP addresses of the client and the server must remain constant for the duration of the connection.

#### *Two Addresses*

The approach that is more feasible is the use of two addresses. The host has its original address, called the **home address,** and a temporary address, called the **care-of address.**

The home address is permanent; it associates the host to its **home network,** the network that is the permanent home of the host. The care-of address is temporary. When a host moves from one network to another, the care-of address changes; it is associated with the **foreign network,** the network to which the host moves. Figure 10.1 shows the concept.

**Figure 10.1** *Home address and care-of address*



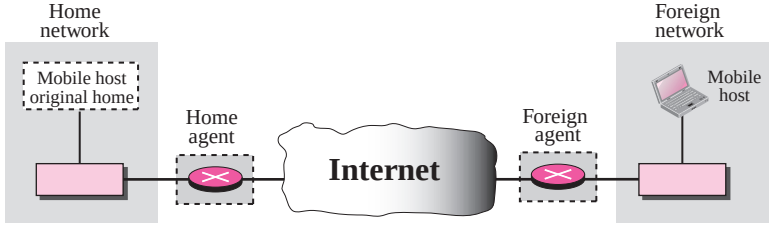

**Mobile IP has two addresses for a mobile host: one home address and one care-of address. The home address is permanent; the care-of address changes as the mobile host moves from one network to another.**

When a mobile host visits a foreign network, it receives its care-of address during the agent discovery and registration phase described later.

## 10.2 AGENTS

To make the change of address transparent to the rest of the Internet requires a **home agent** and a **foreign agent.** Figure 10.2 shows the position of a home agent relative to the home network and a foreign agent relative to the foreign network.

**Figure 10.2** *Home agent and foreign agent*



We have shown the home and the foreign agents as routers, but we need to emphasize that their specific function as an agent is performed in the application layer. In other words, they are both routers and hosts.

### Home Agent

The home agent is usually a router attached to the home network of the mobile host. The home agent acts on behalf of the mobile host when a remote host sends a packet to the mobile host. The home agent receives the packet and sends it to the foreign agent.

### Foreign Agent

The foreign agent is usually a router attached to the foreign network. The foreign agent receives and delivers packets sent by the home agent to the mobile host.

The mobile host can also act as a foreign agent. In other words, the mobile host and the foreign agent can be the same. However, to do this, a mobile host must be able to receive a care-of address by itself, which can be done through the use of DHCP. In addition, the mobile host needs the necessary software to allow it to communicate with the home agent and to have two addresses: its home address and its care-of address. This dual addressing must be transparent to the application programs.

When the mobile host acts as a foreign agent, the care-of address is called a **colocated care-of address.**

**When the mobile host and the foreign agent are the same, the care-of address is called a colocated care-of address.**

The advantage of using a colocated care-of address is that the mobile host can move to any network without worrying about the availability of a foreign agent. The disadvantage is that the mobile host needs extra software to act as its own foreign agent.

## 10.3 THREE PHASES

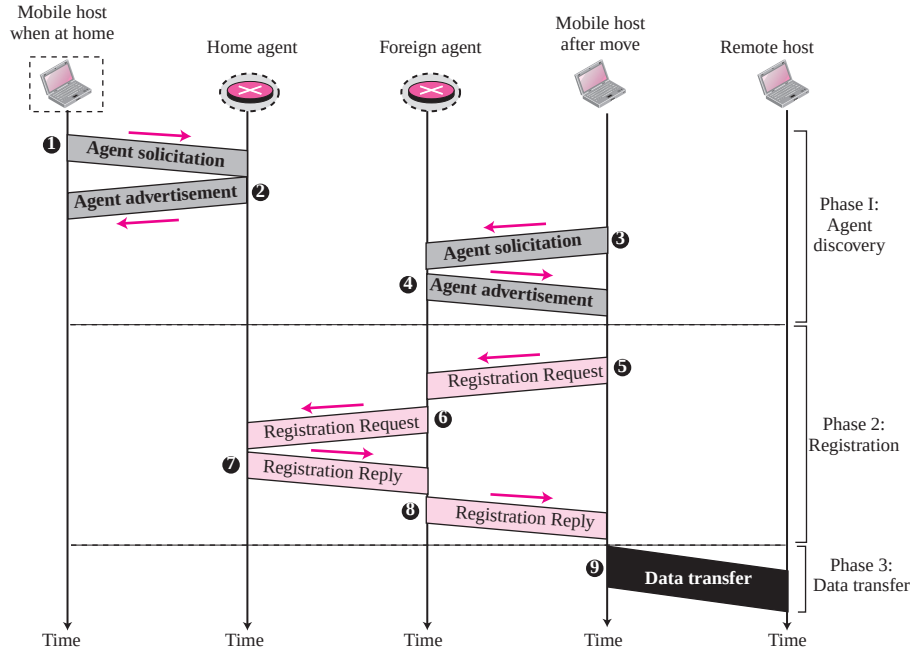
To communicate with a remote host, a mobile host goes through three phases: agent discovery, registration, and data transfer, as shown in Figure 10.3.

The first phase, agent discovery, involves the mobile host, the foreign agent, and the home agent. The second phase, registration, also involves the mobile host and the two agents. Finally, in the third phase, the remote host is also involved. We discuss each phase separately.

### Agent Discovery

The first phase in mobile communication, **agent discovery,** consists of two subphases. A mobile host must discover (learn the address of) a home agent before it leaves its home network. A mobile host must also discover a foreign agent after it has moved to a foreign network. This discovery consists of learning the care-of address as well as the foreign agent's address. The discovery involves two types of messages: advertisement and solicitation.

**Figure 10.3** *Remote host and mobile host communication*

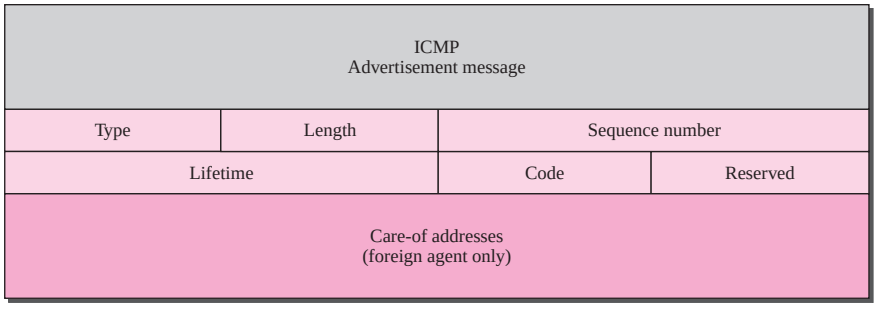



### *Agent Advertisement*

When a router advertises its presence on a network using an ICMP router advertisement, it can append an **agent advertisement** to the packet if it acts as an agent. Figure 10.4 shows how an agent advertisement is piggybacked to the router advertisement packet.

**Mobile IP does not use a new packet type for agent advertisement; it uses the router advertisement packet of ICMP, and appends an agent advertisement message.**

**Figure 10.4** *Agent advertisement*



The field descriptions are as follows:

- **Type.** The 8-bit type field is set to 16.
- **Length.** The 8-bit length field defines the total length of the extension message (not the length of the ICMP advertisement message).
- **Sequence number.** The 16-bit sequence number field holds the message number. The recipient can use the sequence number to determine if a message is lost.
- **Lifetime.** The lifetime field defines the number of seconds that the agent will accept requests. If the value is a string of 1s, the lifetime is infinite.
- **Code.** The code field is an 8-bit flag in which each bit is set (1) or unset (0). The meanings of the bits are shown in Table 10.1.

**Table 10.1** *Code Bits*

| *Bit* | *Meaning* |
|---|---|
| 0 | Registration required. No colocated care-of address. |
| 1 | Agent is busy and does not accept registration at this moment. |
| 2 | Agent acts as a home agent. |
| 3 | Agent acts as a foreign agent. |
| 4 | Agent uses minimal encapsulation. |
| 5 | Agent uses generic routing encapsulation (GRE). |
| 6 | Agent supports header compression. |
| 7 | Unused (0). |

- **Care-of Addresses.** This field contains a list of addresses available for use as care-of addresses. The mobile host can choose one of these addresses. The selection of this care-of address is announced in the registration request. Note that this field is used only by a foreign agent.

### *Agent Solicitation*

When a mobile host has moved to a new network and has not received agent advertisements, it can initiate an **agent solicitation.** It can use the ICMP solicitation message to inform an agent that it needs assistance.

**Mobile IP does not use a new packet type for agent solicitation; it uses the router solicitation packet of ICMP.**

## Registration

The second phase in mobile communication is **registration.** After a mobile host has moved to a foreign network and discovered the foreign agent, it must register. There are four aspects of registration:

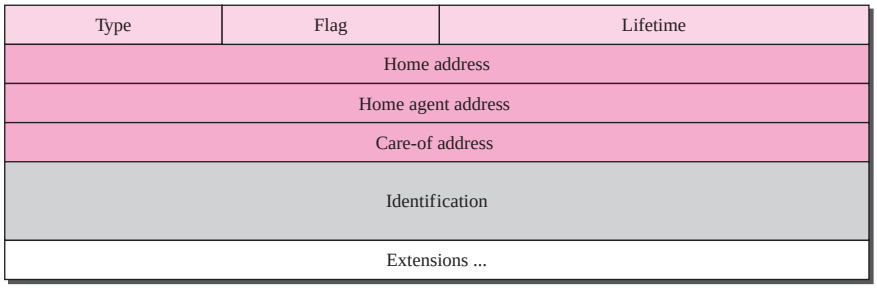
1. The mobile host must register itself with the foreign agent.
2. The mobile host must register itself with its home agent. This is normally done by the foreign agent on behalf of the mobile host.
3. The mobile host must renew registration if it has expired.
4. The mobile host must cancel its registration (deregistration) when it returns home.

### *Request and Reply*

To register with the foreign agent and the home agent, the mobile host uses a **registration request** and a **registration reply** as shown in Figure 10.3.

**Registration Request** A registration request is sent from the mobile host to the foreign agent to register its care-of address and also to announce its home address and home agent address. The foreign agent, after receiving and registering the request, relays the message to the home agent. Note that the home agent now knows the address of the foreign agent because the IP packet that is used for relaying has the IP address of the foreign agent as the source address. Figure 10.5 shows the format of the registration request.

**Figure 10.5** *Registration request format*




The field descriptions are as follows:

- **Type.** The 8-bit type field defines the type of the message. For a request message the value of this field is 1.
- **Flag.** The 8-bit flag field defines forwarding information. The value of each bit can be set or unset. The meaning of each bit is given in Table 10.2.

**Table 10.2** *Registration request flag field bits*

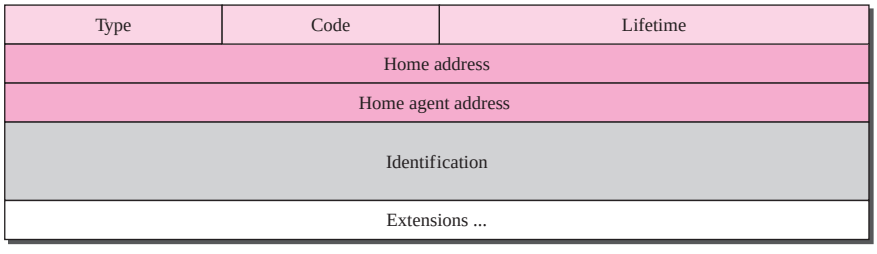
| *Bit* | *Meaning* |
|---|---|
| 0 | Mobile host requests that home agent retain its prior care-of address. |
| 1 | Mobile host requests that home agent tunnel any broadcast message. |
| 2 | Mobile host is using colocated care-of address. |
| 3 | Mobile host requests that home agent use minimal encapsulation. |
| 4 | Mobile host requests generic routing encapsulation (GRE). |
| 5 | Mobile host requests header compression. |
| 6–7 | Reserved bits. |

- **Lifetime.** This field defines the number of seconds the registration is valid. If the field is a string of 0s, the request message is asking for deregistration. If the field is a string of 1s, the lifetime is infinite.
- **Home address.** This field contains the permanent (first) address of the mobile host.

- **Home agent address.** This field contains the address of the home agent.
- **Care-of address.** This field is the temporary (second) address of the mobile host.
- **Identification.** This field contains a 64-bit number that is inserted into the request by the mobile host and repeated in the reply message. It matches a request with a reply.
- **Extensions.** Variable length extensions are used for authentication. They allow a home agent to authenticate the mobile agent. We discuss authentication in Chapter 29.

**Registration Reply** A registration reply is sent from the home agent to the foreign agent and then relayed to the mobile host. The reply confirms or denies the registration request. Figure 10.6 shows the format of the registration reply.

**Figure 10.6** *Registration reply format*




The fields are similar to those of the registration request with the following exceptions. The value of the type field is 3. The code field replaces the flag field and shows the result of the registration request (acceptance or denial). The care-of address field is not needed.

### *Encapsulation*

Registration messages are encapsulated in a UDP user datagram. An agent uses the well-known port 434; a mobile host uses an ephemeral port.

**A registration request or reply is sent by UDP using the well-known port 434.**

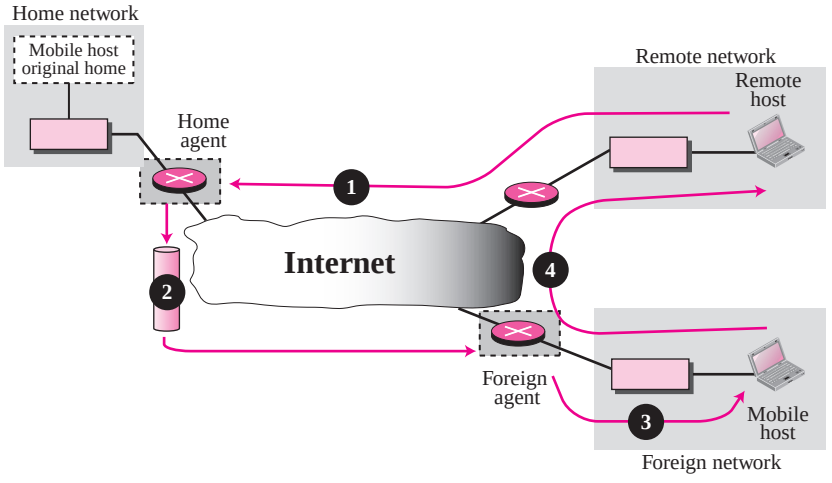
## Data Transfer

After agent discovery and registration, a mobile host can communicate with a remote host. Figure 10.7 shows the idea.

### *From Remote Host to Home Agent*

When a remote host wants to send a packet to the mobile host, it uses its address as the source address and the home address of the mobile host as the destination address. In other words, the remote host sends a packet as though the mobile host is at its home network. The packet, however, is intercepted by the home agent, which pretends it is

**Figure 10.7** *Data transfer*




the mobile host. This is done using the proxy ARP technique discussed in Chapter 8. Path 1 of Figure 10.7 shows this step.

### *From Home Agent to Foreign Agent*

After receiving the packet, the home agent sends the packet to the foreign agent using the tunneling concept discussed in Chapter 30. The home agent encapsulates the whole IP packet inside another IP packet using its address as the source and the foreign agent's address as the destination. Path 2 of Figure 10.7 shows this step.

### *From Foreign Agent to Mobile Host*

When the foreign agent receives the packet, it removes the original packet. However, since the destination address is the home address of the mobile host, the foreign agent consults a registry table to find the care-of address of the mobile host. (Otherwise, the packet would just be sent back to the home network.) The packet is then sent to the care-of address. Path 3 of Figure 10.7 shows this step.

### *From Mobile Host to Remote Host*

When a mobile host wants to send a packet to a remote host (for example, a response to the packet it has received), it sends as it does normally. The mobile host prepares a packet with its home address as the source, and the address of the remote host as the destination. Although the packet comes from the foreign network, it has the home address of the mobile host. Path 4 of Figure 10.7 shows this step.

### *Transparency*

In this data transfer process, the remote host is unaware of any movement by the mobile host. The remote host sends packets using the home address of the mobile host as the destination address; it receives packets that have the home address of the mobile host as

the source address. The movement is totally transparent. The rest of the Internet is not aware of the mobility of the moving host.

**The movement of the mobile host is transparent to the rest of the Internet.**

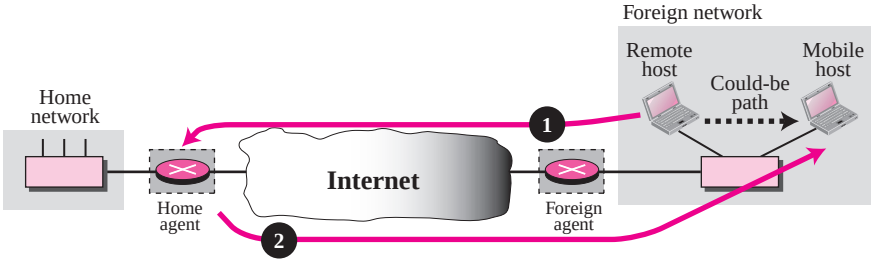
## 10.4 INEFFICIENCY IN MOBILE IP

Communication involving mobile IP can be inefficient. The inefficiency can be severe or moderate. The severe case is called *double crossing* or *2X*. The moderate case is called *triangle routing* or *dog-leg routing*.

### Double Crossing

**Double crossing** occurs when a remote host communicates with a mobile host that has moved to the same network (or site) as the remote host (see Figure 10.8).

**Figure 10.8** *Double crossing*




When the mobile host sends a packet to the remote host, there is no inefficiency; the communication is local. However, when the remote host sends a packet to the mobile host, the packet crosses the Internet twice.

Since a computer usually communicates with other local computers (principle of locality), the inefficiency from double crossing is significant.

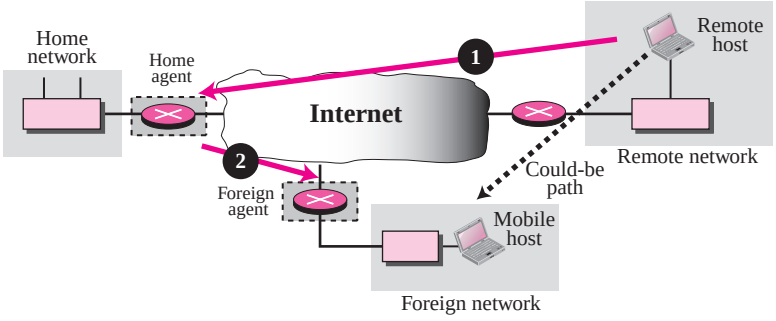
### Triangle Routing

**Triangle routing,** the less severe case, occurs when the remote host communicates with a mobile host that is not attached to the same network (or site) as the mobile host. When the mobile host sends a packet to the remote host, there is no inefficiency. However, when the remote host sends a packet to the mobile host, the packet goes from the remote host to the home agent and then to the mobile host. The packet travels the two sides of a triangle, instead of just one side (see Figure 10.9).

### Solution

One solution to inefficiency is for the remote host to bind the care-of address to the home address of a mobile host. For example, when a home agent receives the first

**Figure 10.9** *Triangle routing*




packet for a mobile host, it forwards the packet to the foreign agent; it could also send an **update binding packet** to the remote host so that future packets to this host could be sent to the care-of address. The remote host can keep this information in a cache.

The problem with this strategy is that the cache entry becomes outdated once the mobile host moves. In this case the home agent needs to send a **warning packet** to the remote host to inform it of the change.

## 10.5 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], [Kur & Ros 08], and [Gar & Vid 04], and [Pet & Dav 03].

### RFCs

Several RFCs in particular discuss mobile IP: RFC 1701, RFC 2003, RFC 2004, RFC 3024, RFC 3344, and RFC 3775.

## 10.6 KEY TERMS

agent advertisement
agent discovery
agent solicitation
care-of address
colocated care-of address
double crossing
foreign agent
foreign network
home address
home agent
home network
mobile host

registration
registration reply
registration request
stationary host
triangle routing
update binding packet
warning packet

## 10.7 SUMMARY

- Mobile IP, designed for mobile communication, is an enhanced version of the Internetworking Protocol (IP). A mobile host has a home address on its home network and a care-of address on its foreign network. When the mobile host is on a foreign network, a home agent relays messages (for the mobile host) to a foreign agent. A foreign agent sends relayed messages to a mobile host.
- A mobile host on its home network learns the address of a home agent through a process called agent discovery. A mobile host on a foreign network learns the address of a foreign agent through agent discovery or agent solicitation.
- A mobile host on a foreign network must register itself with both the home and foreign agents.
- A message from a remote host goes from the remote host to the home agent, to the foreign agent, and then to the mobile host.
- Mobile communication can be inefficient due to the extra distance a message must travel. Double crossing and triangle routing are two instances of inefficient routing.

## 10.8 PRACTICE SET

### Exercises

1. Is registration required if the mobile host acts as a foreign agent? Explain your answer.
2. Redraw Figure 10.7 if the mobile host acts as a foreign agent.
3. Create a home agent advertisement message using 1456 as the sequence number and a lifetime of 3 hours. Select your own values for the bits in the code field. Calculate and insert the value for the length field.
4. Create a foreign agent advertisement message using 1672 as the sequence number and a lifetime of 4 hours. Select your own values for the bits in the code field. Use at least three care-of addresses of your choice. Calculate and insert the value for the length field.
5. Discuss how the ICMP router solicitation message can also be used for agent solicitation. Why are there no extra fields?
6. Which protocol is the carrier of the agent advertisement and solicitation messages?
7. Show the encapsulation of the advertisement message in Exercise 3 in an IP datagram. What is the value for the protocol field?

8. Explain why the registration request and reply are not directly encapsulated in an IP datagram. Why is there a need for the UDP user datagram?
9. We have the following information shown below. Show the contents of the IP datagram header sent from the remote host to the home agent.

**Mobile host home address: 130.45.6.7/16**
**Mobile host care-of address: 14.56.8.9/8**
**Remote host address: 200.4.7.14/24**
**Home agent address: 130.45.10.20/16**
**Foreign agent address: 14.67.34.6/8**

10. Using the information in Exercise 9, show the contents of the IP datagram sent by the home agent to the foreign agent. Use tunneling.
11. Using the information in Exercise 9, show the contents of the IP datagram sent by the foreign agent to the mobile host.
12. Using the information in Exercise 9, show the contents of the IP datagram sent by the mobile host to the remote host.
13. What type of inefficiency do we have in Exercise 9? Explain your answer.

## Research Activities

14. We mentioned that registration messages are encapsulated in UDP. Find why UDP is chosen instead of TCP.
15. Find how frequently an agent advertisement is sent.
16. Find the different types of authentication needed in mobile IP.
17. Find the role of multicasting in mobile IP.

CHAPTER

# 11

# *Unicast Routing Protocols (RIP, OSPF, and BGP)*

As we have discussed in some previous chapters, unicast communication means communication between one sender and one receiver, a one-to-one communication. In this chapter, we discuss how the routers create their routing tables to support unicast communication. We show how the Internet is divided into administrative areas known as autonomous systems to efficiently handle the exchange of routing information. We then explain two dominant routing protocols used inside an autonomous system and one routing protocol used for exchange of routing information between autonomous systems.

## OBJECTIVES

*The chapter has several objectives:*

- To introduce the idea of autonomous systems (ASs) that divide the Internet into smaller administrative regions for the purpose of exchanging routing information.
- To discuss the idea of distance vector routing as the first intra-AS routing method and how it uses the Bellman-Ford algorithm to update routing tables.
- To discuss how the Routing Information Protocol (RIP) is used to implement the idea of distance vector routing in the Internet.
- To discuss the idea of link state routing as the second intra-AS routing method and how it uses Dijkstra algorithm to update the routing tables.
- To discuss how Open Shortest Path First (OSPF) is used to implement the idea of link state routing in the Internet.
- To discuss the idea of path vector routing as the dominant inter-AS routing method and explain the concept of policy routing.
- To discuss how Border Gateway Protocol (BGP) is used to implement the idea of path vector routing in the Internet.

## 11.1 INTRODUCTION

An internet is a combination of networks connected by routers. When a datagram goes from a source to a destination, it will probably pass through many routers until it reaches the router attached to the destination network.

### Cost or Metric

A router receives a packet from a network and passes it to another network. A router is usually attached to several networks. When it receives a packet, to which network should it pass the packet? The decision is based on optimization: Which of the available pathways is the optimum pathway? What is the definition of the term *optimum*?

One approach is to assign a **cost** for passing through a network. We call this cost a **metric.** High cost can be thought of as something *bad*; low cost can be thought of something *good*. For example, if we want to maximize the throughput in a network, the high throughput means low cost and the low throughput means high cost. As another example, if we want to minimize the delay, low delay is low cost and high delay is high cost.

### Static versus Dynamic Routing Tables

A routing table can be either static or dynamic. A *static table* is one with manual entries. A *dynamic table,* on the other hand, is one that is updated automatically when there is a change somewhere in the internet. Today, an internet needs dynamic routing tables. The tables need to be updated as soon as there is a change in the internet. For instance, they need to be updated when a link is down, and they need to be updated whenever a better route has been found.

### Routing Protocol

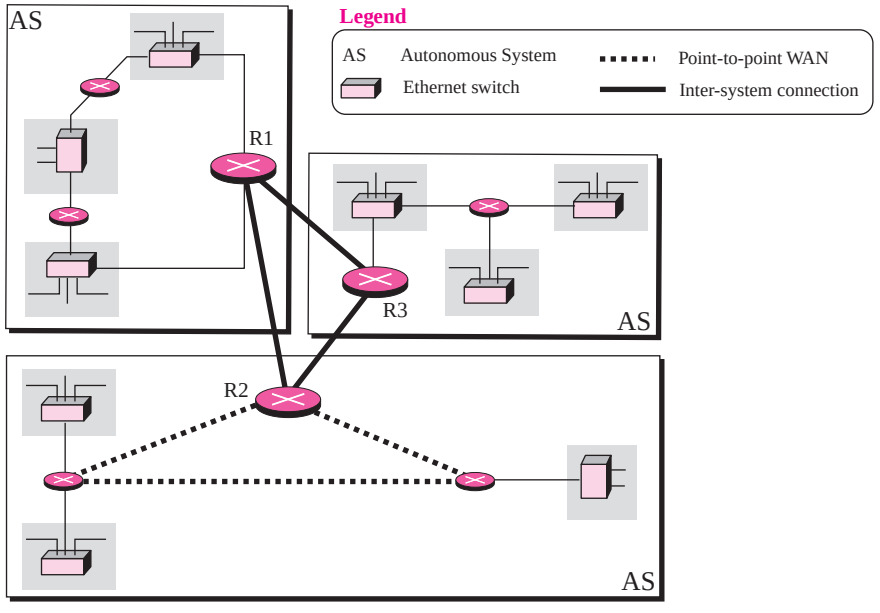
Routing protocols have been created in response to the demand for dynamic routing tables. A routing protocol is a combination of rules and procedures that lets routers in the internet inform each other of changes. It allows routers to share whatever they know about the internet or their neighborhood. The sharing of information allows a router in San Francisco to know about the failure of a network in Texas. The routing protocols also include procedures for combining information received from other routers.

Routing protocols can be either an *interior protocol* or an *exterior protocol*. An interior protocol handles *intradomain routing*; an exterior protocol handles *inter-domain routing*. We start the next section with defining these terms.

## 11.2 INTRA- AND INTER-DOMAIN ROUTING

Today, an internet can be so large that one routing protocol cannot handle the task of updating the routing tables of all routers. For this reason, an internet is divided into autonomous systems. An **autonomous system (AS)** is a group of networks and routers under the authority of a single administration. Routing inside an autonomous system is referred to as *intra-domain routing.* Routing between autonomous systems is referred to as *inter-domain routing*. Each autonomous system can choose one or more intradomain routing protocols to handle routing inside the autonomous system. However, only one interdomain routing protocol handles routing between autonomous systems. See Figure 11.1.

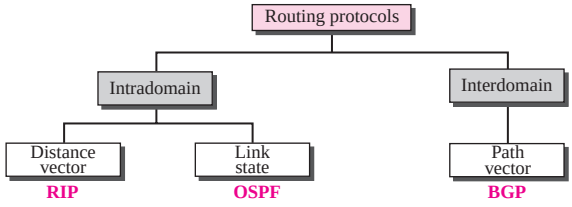
**Figure 11.1** *Autonomous systems*




Several intra-domain and inter-domain routing protocols are in use. In this chapter, we cover only the most popular ones. We discuss two intra-domain routing protocols: distance vector and link state. We also introduce one inter-domain routing protocol: path vector (see Figure 11.2).

Routing Information Protocol (RIP) is the implementation of the distance vector protocol. Open Shortest Path First (OSPF) is the implementation of the link state protocol. Border Gateway Protocol (BGP) is the implementation of the path vector protocol. RIP and OSPF are interior routing protocols; BGP is an exterior routing protocol.

**Figure 11.2** *Popular routing protocols*



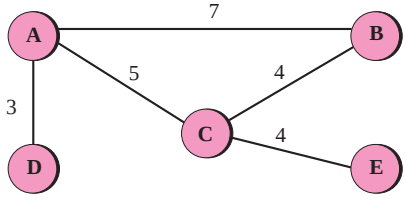

## 11.3 DISTANCE VECTOR ROUTING

We first discuss **distance vector routing.** This method sees an AS, with all routers and networks, as a *graph*, a set of nodes and lines (edges) connecting the nodes. A router can normally be represented by a node and a network by a link connecting two nodes, although other representations are also possible. The graph theory used an algorithm called Bellman-Ford (also called Ford-Fulkerson) for a while to find the shortest path between nodes in a graph given the distance between nodes. We first discuss this algorithm before we see how it can be modified to be used for updating routing tables in a distance vector routing.

### Bellman-Ford Algorithm

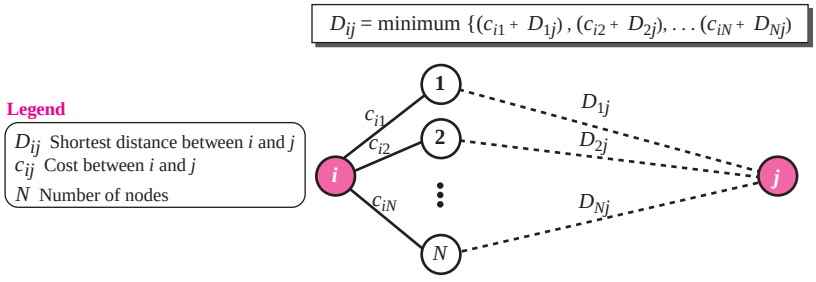
Let us briefly discuss the **Bellman-Ford algorithm.** The algorithm can be used in many applications in graph theory. If we know the cost between each pair of nodes, we can use the algorithm to find the least cost (shortest path) between any two nodes. Figure 11.3 shows a map with nodes and lines. The cost of each line is given over the line; the algorithm can find the least cost between any two nodes. For example, if the nodes represent cities and the lines represent roads connecting them, the graph can find the shortest distance between any two cities.

**Figure 11.3** *A graph for the Bellman-Ford algorithm*




The algorithm is based on the fact that if all neighbors of node *i* know the shortest distance to node *j*, then the shortest distance between node *i* and *j* can be found by adding the distance between node *i* and each neighbor to the neighbor's shortest distance to node *j* and then select the minimum, as shown in Figure 11.4.

**Figure 11.4** *The fact behind Bellman-Ford algorithm*




Although the principle of the Bellman-Ford algorithm is very simple and intuitive, the algorithm itself looks circular. How have the neighbors calculated the shortest path to the destination? To solve the problem, we use iteration. We create a shortest distance table (vector) for each node using the following steps:

1. The shortest distance and the cost between a node and itself is initialized to 0.
2. The shortest distance between a node and any other node is set to infinity. The cost between a node and any other node should be given (can be infinity if the nodes are not connected).
3. The algorithm repeat as shown in Figure 11.4 until there is no more change in the shortest distance vector.

Table 11.1 shows the algorithm in pseudocode.

**Table 11.1** *Bellman-Ford Algorithm*

```
1  Bellman_Ford ( )
2  {
3     // Initialization
4     for (i = 1 to N; for j = 1 to N)
5     {
6        if(i == j)  Dij = 0   cij = 0
7        else        Dij = ∞   cij = cost between i and j
8     }
9     // Updating
10    repeat
11    {
12       for (i = 1 to N; for j = 1 to N)
13       {
14          Dij  ← minimum [(ci1 + D1j)  ... (ciN + DNj)]
15       } // end for
16    } until (there was no change in previous iteration)
17 } // end Bellman-Ford
```

## Distance Vector Routing Algorithm

The Bellman-Ford algorithm can be very well applied to a map of roads between cities because we can have all of the initial information about each node at the same place. We can enter this information into the computer and let the computer hold the intermediate results and create the final vectors for each node to be printed. In other words, the algorithm is designed to create the result *synchronously*. If we want to use the algorithm for creating the routing table for routers in an AS, we need to change the algorithm:

1. In distance vector routing, the cost is normally hop counts (how many networks are passed before reaching the destination). So the cost between any two neighbors is set to 1.
2. Each router needs to update its routing table *asynchronously*, whenever it has received some information from its neighbors. In other words, each router executes part of the whole algorithm in the Bellman-Ford algorithm. Processing is *distributive*.
3. After a router has updated its routing table, it should send the result to its neighbors so that they can also update their routing table.
4. Each router should keep at least three pieces of information for each route: destination network, the cost, and the next hop. We refer to the whole routing table as Table, to the row $i$ in the table as $\text{Table}_i$, to the three columns in row $i$ as $\text{Table}_i$.dest, $\text{Table}_i$.cost, and $\text{Table}_i$.next.
5. We refer to information about each route received from a neighbor as R (record), which has only two pieces of information: R.dest and R.cost. The next hop is not included in the received record because it is the source address of the sender.

Table 11.2 shows the algorithm in pseudocode.

**Table 11.2** *Distance Vector Algorithm Used by Each Router*

```
1  Distance_Vector_Algorithm ( )
2  {
3      // At startup
4     for (i = 1 to N)          // N is number of ports
5     {
6         Table_i.dest = address of the attached network
7         Table_i.cost = 1
8         Table_i.next = —       // Means at home
9         Send a record R about each row to each neighbor
10    } // end for loop
11
12      // Updating
13    repeat (forever)
14    {
15        Wait for arrival of a record R from a neighbor
16        Update (R, T)          // Call update module
17        for (i = 1 to N)       // N is the current table size
```

**Table 11.2** *Distance Vector Algorithm Used by Each Router (continued)*

```
18        {
19              Send a record R about each row to each neighbor
20        }
21       }  // end repeat
22
23  }  // end Distance_Vector
24  Update (R, T)          // Update module
25  {
26      Search T for a destination matching the one in R
27      if (destination is found in row i)
28      {
29          if (R.cost + 1 < T_i.cost    or    R.next == T_i.next)
30          {
31                T_i.cost = R.cost + 1
32                T_i.next = Address of sending router
33          }
34       else  discard  the  record     // No change is needed
35       }
36       else
37        // Insert the new router
38       {
39           T_N+1.dest = R.dest
40           T_N+1.cost = R.cost + 1
41           T_N+1.next = Address of sending router
42           Sort the table according to destination address
43       }
44  }  // end of Update module
```

Lines 4 to 10 show the initialization at the start-up. The router creates a preliminary routing table that can only be used to route packets to the networks directly attached to its interfaces. For each entry in the routing table, it sends a record with only two fields: destination address and the cost to each neighbor.

The router updates itself whenever it receives a record from a neighbor. After each update, the route sends a record for each entry in the routing table to its neighbors to let them also update themselves.

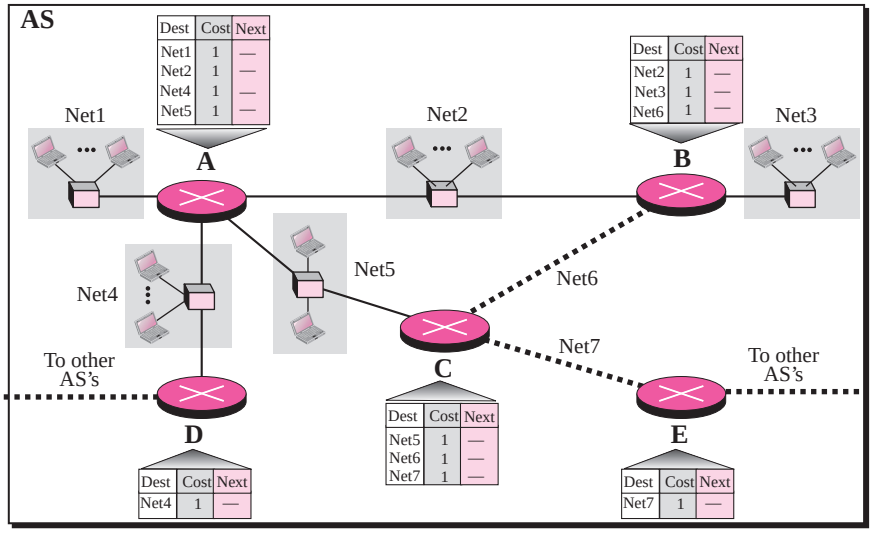
Lines 23 to 47 give the details of updating process. When a record arrives, the router searches for the destination address in the routing table.

1. If the corresponding entry is found, two cases need to be checked and the route information should be changed.
   a. If the record cost plus 1 is smaller than the corresponding cost in the table, it means the neighbors have found a better route to that destination.
   b. If the next hop is the same, it means some change has happened in some part of the network. For example, suppose a neighbor has previously advertised a route to a destination with cost 3, but now there is no path between this neighbor and that destination. The neighbor advertises this destination with cost value infinity. The receiving router must not ignore this value even though its old entry is smaller. The old route does not exist any more.
2. If the entry is not in the table, the router adds it to the table and sorts the table according to the destination address.

### Example 11.1

Figure 11.5 shows the initial routing table for an AS. Note that the figure does not mean that all routing tables have been created at the same time; each router creates its own routing table when it is booted.

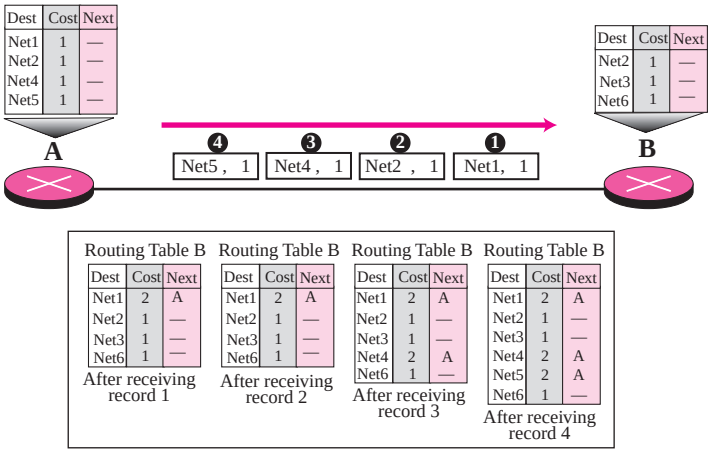
**Figure 11.5** *Example 11.1*




### Example 11.2

Now assume router A sends four records to its neighbors, routers B, D, and C. Figure 11.6 shows the changes in B's routing table when it receives these records. We leave the changes in the routing tables of other neighbors as exercise.

**Figure 11.6** *Example 11.2*




**a.** When router B receives record 1, it searches its routing table for the route to net1, and since it is not found there, it adds one hop to the cost (distance between B and A) and adds it to the table with the next hop to be A.

**b.** When router B receives record 2, it searches its routing table and finds the destination net2 there. However, since the announced cost plus 1 is larger than the cost in the table, the record is discarded.

**c.** When router B receives record 3, it searches its router, and since Net4 is not found, it is added to the table.

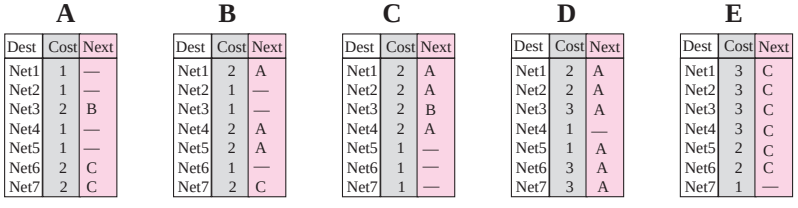
**d.** When router B receives record 4, it searches its router, and since Net5 is not found, it is added to the table.

Now router B has more information, but it is not complete. Router B does not even know that net7 exists. More updating is required.

## Example 11.3

Figure 11.7 shows the final routing tables for routers in Figure 11.5.

**Figure 11.7** *Example 11.3*



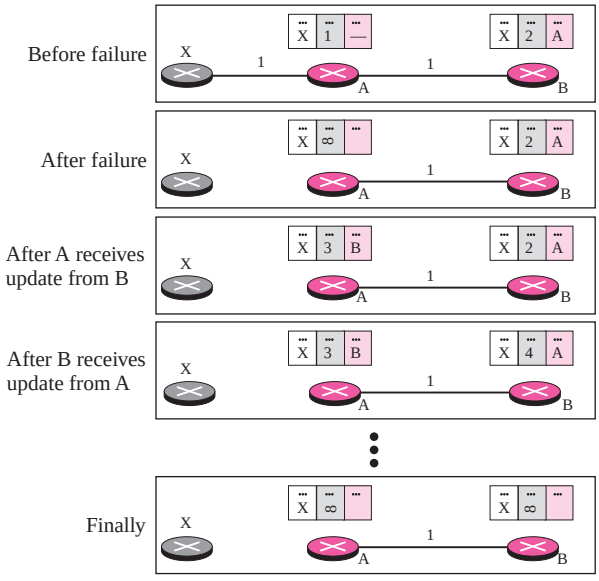
## Count to Infinity

A problem with distance vector routing is that any decrease in cost (good news) propagates quickly, but any increase in cost (bad news) propagates slowly. For a routing protocol to work properly, if a link is broken (cost becomes infinity), every other router should be aware of it immediately, but in distance vector routing, this takes some time. The problem is referred to as **count to infinity.** It takes several updates before the cost for a broken link is recorded as infinity by all routers.

### *Two-Node Loop*

One example of count to infinity is the two-node loop problem. To understand the problem, let us look at the scenario depicted in Figure 11.8.

**Figure 11.8** *Two-node instability*




The figure shows a system with three nodes. We have shown only the portions of the routing table needed for our discussion. At the beginning, both nodes A and B know how to reach node X. But suddenly, the link between A and X fails. Node A changes its table. If A can send its table to B immediately, everything is fine. However, the system becomes unstable if B sends its routing table to A before receiving A's routing table. Node A receives the update and, assuming that B has found a way to reach X, immediately updates its routing table. Now A sends its new update to B. Now B thinks that something has been changed around A and updates its routing table. The cost of reaching X increases gradually until it reaches infinity. At this moment, both A and B know that X cannot be reached. However, during this time the system is not stable. Node A thinks that the route to X is via B; node B thinks that the route to X is via A. If A

receives a packet destined for X, it goes to B and then comes back to A. Similarly, if B receives a packet destined for X, it goes to A and comes back to B. Packets bounce between A and B, creating a two-node loop problem. A few solutions have been proposed for instability of this kind.

**Defining Infinity** The first obvious solution is to redefine infinity to a smaller number, such as 16. For our previous scenario, the system will be stable in fewer updates. As a matter of fact, most implementations of the Distance Vector Protocol define 16 as infinity. However, this means that distance vector cannot be used in large systems. The size of the network, in each direction, can not exceed 15 hops.

**Split Horizon** Another solution is called **split horizon.** In this strategy, instead of flooding the table through each interface, each node sends only part of its table through each interface. If, according to its table, node B thinks that the optimum route to reach X is via A, it does not need to advertise this piece of information to A; the information has come from A (A already knows). Taking information from node A, modifying it, and sending it back to node A is what creates the confusion. In our scenario, node B eliminates the last line of its routing table before it sends it to A. In this case, node A keeps the value of infinity as the distance to X. Later, when node A sends its routing table to B, node B also corrects its routing table. The system becomes stable after the first update: both node A and B know that X is not reachable.

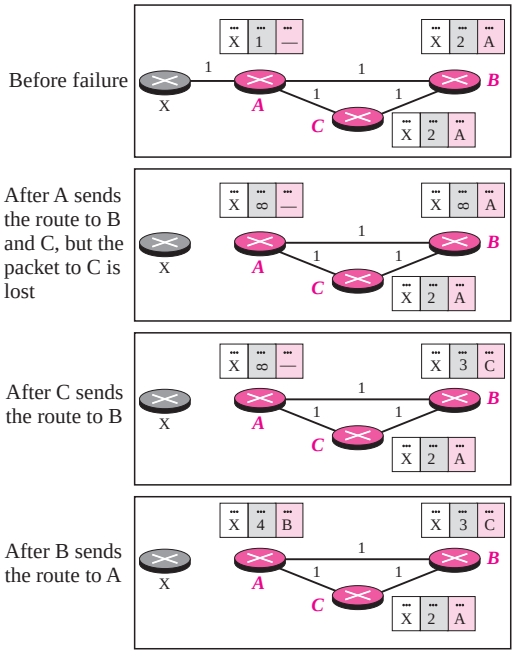
**Split Horizon and Poison Reverse** Using the split horizon strategy has one drawback. Normally, the Distance Vector Protocol uses a timer, and if there is no news about a route, the node deletes the route from its table. When node B in the previous scenario eliminates the route to X from its advertisement to A, node A cannot guess that this is due to the split horizon strategy (the source of information was A) or because B has not received any news about X recently. The split horizon strategy can be combined with the **poison reverse** strategy. Node B can still advertise the value for X, but if the source of information is A, it can replace the distance with infinity as a warning: "Do not use this value; what I know about this route comes from you."

### *Three-Node Instability*

The two-node instability can be avoided using split horizon combined with poison reverse. However, if the instability is between three nodes, stability cannot be guaranteed. Figure 11.9 shows the scenario.

Suppose, after finding that X is not reachable, node A sends a packet to B and C to inform them of the situation. Node B immediately updates its table, but the packet to C is lost in the network and never reaches C. Node C remains in the dark and still thinks that there is a route to X via A with a distance of 2. After a while, node C sends its routing table to B, which includes the route to X. Node B is totally fooled here. It receives information on the route to X from C, and according to the algorithm, it updates its table showing the route to X via C with a cost of 3. This information has come from C, not from A, so after awhile node B may advertise this route to A. Now A is fooled and updates its table to show that A can reach X via B with a cost of 4. Of course, the loop continues; now A advertises the route to X to C, with increased cost, but not to B. C then advertises the route to B with an increased cost. B does the same to A. And so on. The loop stops when the cost in each node reaches infinity.

**Figure 11.9** *Three-node instability*




## 11.4 RIP

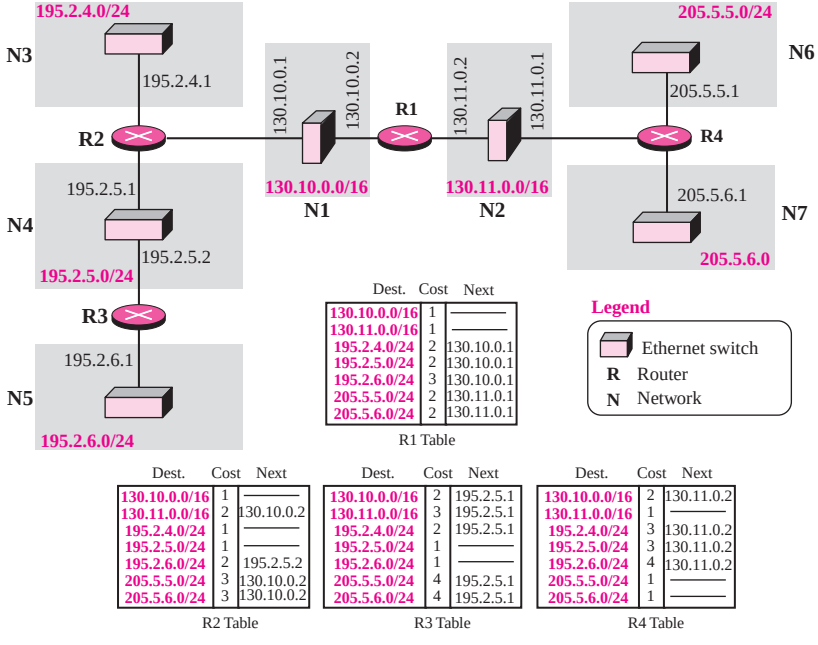
The **Routing Information Protocol (RIP)** is an intradomain (interior) routing protocol used inside an autonomous system. It is a very simple protocol based on distance vector routing. RIP implements distance vector routing directly with some considerations:

1. In an autonomous system, we are dealing with routers and networks (links), what was described as a node.
2. The destination in a routing table is a network, which means the first column defines a network address.
3. The metric used by RIP is very simple; the distance is defined as the number of links (networks) that have to be used to reach the destination. For this reason, the metric in RIP is called a **hop count.**
4. Infinity is defined as 16, which means that any route in an autonomous system using RIP cannot have more than 15 hops.
5. The next node column defines the address of the router to which the packet is to be sent to reach its destination.

Figure 11.10 shows an autonomous system with seven networks and four routers. The table of each router is also shown. Let us look at the routing table for R1. The table has seven entries to show how to reach each network in the autonomous system. Router R1 is

directly connected to networks 130.10.0.0 and 130.11.0.0, which means that there are no next hop entries for these two networks. To send a packet to one of the three networks at the far left, router R1 needs to deliver the packet to R2. The next node entry for these three networks is the interface of router R2 with IP address 130.10.0.1. To send a packet to the two networks at the far right, router R1 needs to send the packet to the interface of router R4 with IP address 130.11.0.1. The other tables can be explained similarly.

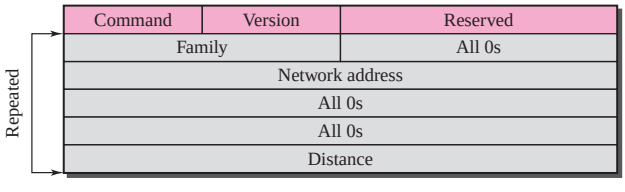
**Figure 11.10** *Example of a domain using RIP*




## RIP Message Format

The format of the RIP message is shown in Figure 11.11.

**Figure 11.11** *RIP message format*



- **Command.** This 8-bit field specifies the type of message: request (1) or response (2).
- **Version.** This 8-bit field defines the version. In this book we use version 1, but at the end of this section, we give some new features of version 2.
- **Family.** This 16-bit field defines the family of the protocol used. For TCP/IP the value is 2.
- **Network address.** The address field defines the address of the destination network. RIP has allocated 14 bytes for this field to be applicable to any protocol. However, IP currently uses only 4 bytes. The rest of the address is filled with 0s.
- **Distance.** This 32-bit field defines the hop count (cost) from the advertising router to the destination network.

Note that part of the message is repeated for each destination network. We refer to this as an *entry.*

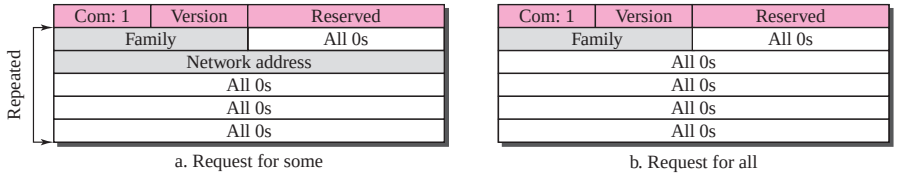
## Requests and Responses

RIP has two types of messages: request and response.

### *Request*

A request message is sent by a router that has just come up or by a router that has some time-out entries. A request can ask about specific entries or all entries (see Figure 11.12).

**Figure 11.12** *Request messages*




### *Response*

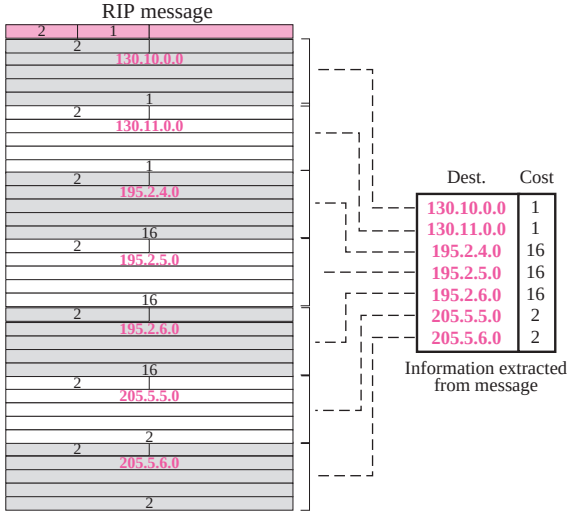
A response can be either solicited or unsolicited. A *solicited response* is sent only in answer to a request. It contains information about the destination specified in the corresponding request. An *unsolicited response,* on the other hand, is sent periodically, every 30 seconds or when there is a change in the routing table. The response is sometimes called an update packet. Figure 11.11 shows the response message format.

### Example 11.4

Figure 11.13 shows the update message sent from router R1 to router R2 in Figure 11.10. The message is sent out of interface 130.10.0.2.

The message is prepared with the combination of split horizon and poison reverse strategy in mind. Router R1 has obtained information about networks 195.2.4.0, 195.2.5.0, and 195.2.6.0 from router R2. When R1 sends an update message to R2, it replaces the actual value of the hop counts for these three networks with 16 (infinity) to prevent any confusion for R2. The figure also shows the table extracted from the message. Router R2 uses the source address of the IP datagram

**Figure 11.13** *Solution to Example 11.4*

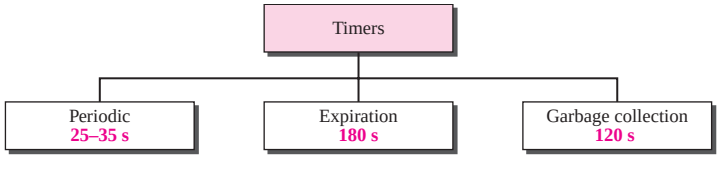



carrying the RIP message from R1 (130.10.02) as the next hop address. Router R2 also increments each hop count by 1 because the values in the message are relative to R1, not R2.

## Timers in RIP

RIP uses three timers to support its operation (see Figure 11.14). The periodic timer controls the sending of messages, the expiration timer governs the validity of a route, and the garbage collection timer advertises the failure of a route.

**Figure 11.14** *RIP timers*




### *Periodic Timer*

The **periodic timer** controls the advertising of regular update messages. Although the protocol specifies that this timer must be set to 30 s, the working model uses a random number between 25 and 35 s. This is to prevent any possible synchronization and therefore overload on an internet if routers update simultaneously.

Each router has one periodic timer that is randomly set to a number between 25 and 35. It counts down; when zero is reached, the update message is sent, and the timer is randomly set once again.

### *Expiration Timer*

The **expiration timer** governs the validity of a route. When a router receives update information for a route, the expiration timer is set to 180 s for that particular route. Every time a new update for the route is received, the timer is reset. In normal situations this occurs every 30 s. However, if there is a problem on an internet and no update is received within the allotted 180 s, the route is considered expired and the hop count of the route is set to 16, which means the destination is unreachable. Every route has its own expiration timer.

### *Garbage Collection Timer*

When the information about a route becomes invalid, the router does not immediately purge that route from its table. Instead, it continues to advertise the route with a metric value of 16. At the same time, a timer called the **garbage collection timer** is set to 120 s for that route. When the count reaches zero, the route is purged from the table. This timer allows neighbors to become aware of the invalidity of a route prior to purging.

### Example 11.5

A routing table has 20 entries. It does not receive information about five routes for 200 s. How many timers are running at this time?

**Solution**

The 21 timers are listed below:

Periodic timer: 1

Expiration timer: $20 - 5 = 15$

Garbage collection timer: 5

## RIP Version 2

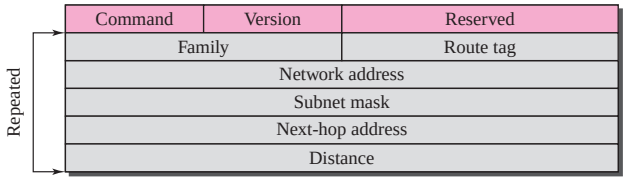
RIP version 2 was designed to overcome some of the shortcomings of version 1. The designers of version 2 have not augmented the length of the message for each entry. They have only replaced those fields in version 1 that were filled with 0s for the TCP/IP protocol with some new fields.

### *Message Format*

Figure 11.15 shows the format of a RIP version 2 message. The new fields of this message are as follows:

- **Route tag.** This field carries information such as the autonomous system number. It can be used to enable RIP to receive information from an interdomain routing protocol.
- **Subnet mask.** This is a 4-byte field that carries the subnet mask (or prefix). This means that RIP2 supports classless addressing and CIDR.
- **Next-hop address.** This field shows the address of the next hop. This is particularly useful if two autonomous systems share a network (a backbone, for example). Then the message can define the router, in the same autonomous system or another autonomous system, to which the packet next goes.

**Figure 11.15** *RIP version 2 format*



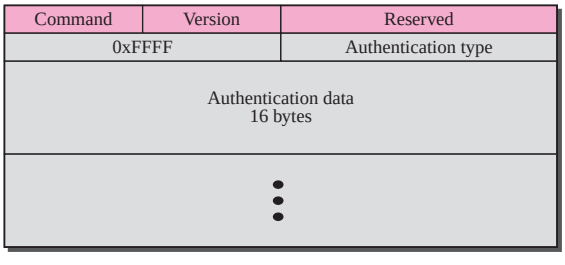

### *Classless Addressing*

Probably the most important difference between the two versions of RIP is classful versus classless addressing. RIPv1 uses classful addressing. The only entry in the message format is the network address (with a default mask). RIPv2 adds one field for the subnet mask, which can be used to define a network prefix length. This means that in this version, we can use classless addressing. A group of networks can be combined into one prefix and advertised collectively, as we saw in Chapters 5 and 6.

### *Authentication*

Authentication is added to protect the message against unauthorized advertisement. No new fields are added to the packet; instead, the first entry of the message is set aside for authentication information. To indicate that the entry is authentication information and not routing information, the value of $FFFF_{16}$ is entered in the family field (see Figure 11.16). The second field, the authentication type, defines the protocol used for authentication, and the third field contains the actual authentication data.

**Figure 11.16** *Authentication*




### *Multicasting*

Version 1 of RIP uses broadcasting to send RIP messages to every neighbor. In this way, all the routers on the network receive the packets, as well as the hosts. RIP version 2, on the other hand, uses the all-router multicast address to send the RIP messages only to RIP routers in the network.

### Encapsulation

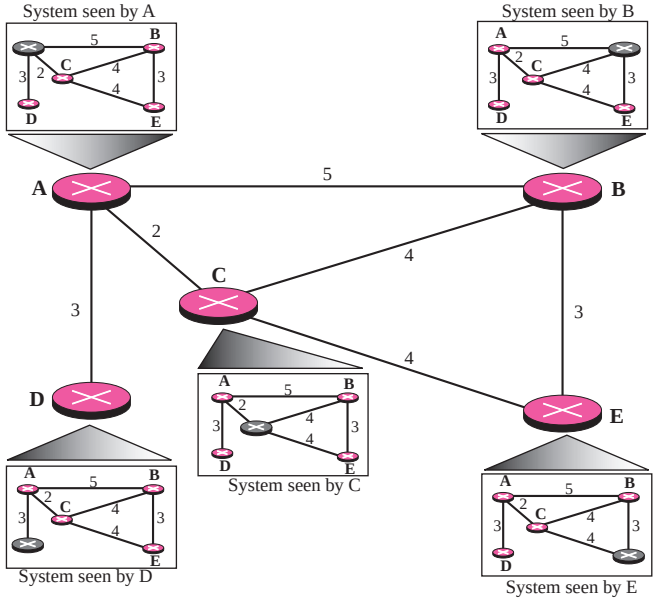
RIP messages are encapsulated in UDP user datagrams. A RIP message does not include a field that indicates the length of the message. This can be determined from the UDP packet. The well-known port assigned to RIP in UDP is port 520.

**RIP uses the services of UDP on well-known port 520.**

## 11.5 LINK STATE ROUTING

Link state routing has a different philosophy from that of distance vector routing. In link state routing, if each node in the domain has the entire topology of the domain—the list of nodes and links, how they are connected including the type, cost (metric), and the condition of the links (up or down)—the node can use the **Dijkstra algorithm** to build a routing table. Figure 11.17 shows the concept.

**Figure 11.17** *Concept of link state routing*

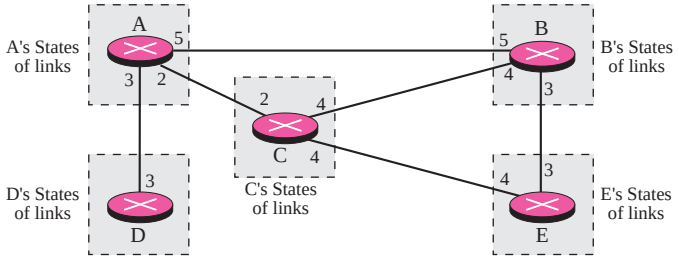



The figure shows a simple domain with five nodes. Each node uses the same topology to create a routing table, but the routing table for each node is unique because the calculations are based on different interpretations of the topology. This is analogous to a city map. Two persons in two different cities may have the same map, but each needs to take a different route to reach his destination.

The topology must be dynamic, representing the latest situation of each node and each link. If there are changes in any point in the network (a link is down, for example), the topology must be updated for each node.

How can a common topology be dynamic and stored in each node? No node can know the topology at the beginning or after a change somewhere in the network. Link state routing is based on the assumption that, although the global knowledge about the topology is not clear, each node has partial knowledge: it knows the state (type, condition, and cost) of its links. In other words, the whole topology can be compiled from the partial knowledge of each node. Figure 11.18 shows the same domain as in the previous figure, indicating the part of the knowledge belonging to each node.

**Figure 11.18** *Link state knowledge*




Node A knows that it is connected to node B with metric 5, to node C with metric 2, and to node D with metric 3. Node C knows that it is connected to node A with metric 2, to node B with metric 4, and to node E with metric 4. Node D knows that it is connected only to node A with metric 3. And so on. Although there is an overlap in the knowledge, the overlap guarantees the creation of a common topology: a picture of the whole domain for each node.

## Building Routing Tables

In **link state routing,** four sets of actions are required to ensure that each node has the routing table showing the least-cost node to every other node.

1. Creation of the states of the links by each node, called the link state packet or LSP.
2. Dissemination of LSPs to every other router, called **flooding,** in an efficient and reliable way.
3. Formation of a shortest path tree for each node.
4. Calculation of a routing table based on the shortest path tree.

### *Creation of Link State Packet (LSP)*

A link state packet (LSP) can carry a large amount of information. For the moment, however, we assume that it carries a minimum amount of data: the node identity, the list of links, a sequence number, and age. The first two, node identity and the list of links, are needed to make the topology. The third, sequence number, facilitates flooding and

distinguishes new LSPs from old ones. The fourth, age, prevents old LSPs from remaining in the domain for a long time. LSPs are generated on two occasions:

1. *When there is a change in the topology of the domain.* Triggering of LSP dissemination is the main way of quickly informing any node in the domain to update its topology.
2. *On a periodic basis.* The period in this case is much longer compared to distance vector routing. As a matter of fact, there is no actual need for this type of LSP dissemination. It is done to ensure that old information is removed from the domain. The timer set for periodic dissemination is normally in the range of 60 minutes or 2 hours based on the implementation. A longer period ensures that flooding does not create too much traffic on the network.

### *Flooding of LSPs*

After a node has prepared an LSP, it must be disseminated to all other nodes, not only to its neighbors. The process is called flooding and based on the following:

1. The creating node sends a copy of the LSP out of each interface.
2. A node that receives an LSP compares it with the copy it may already have. If the newly arrived LSP is older than the one it has (found by checking the sequence number), it discards the LSP. If it is newer, the node does the following:
   a. It discards the old LSP and keeps the new one.
   b. It sends a copy of it out of each interface except the one from which the packet arrived. This guarantees that flooding stops somewhere in the domain (where a node has only one interface).

### *Formation of Shortest Path Tree: Dijkstra Algorithm*

After receiving all LSPs, each node will have a copy of the whole topology. However, the topology is not sufficient to find the shortest path to every other node; a **shortest path tree** is needed.

A tree is a graph of nodes and links; one node is called the root. All other nodes can be reached from the root through only one single route. A shortest path tree is a tree in which the path between the root and every other node is the shortest. What we need for each node is a shortest path tree with that node as the root. The **Dijkstra algorithm** is used to create a shortest path tree from a given graph. The algorithm uses the following steps:

1. **Initialization:** Select the node as the root of the tree and add it to the *path*. Set the shortest distances for all the root's neighbors to the cost between the root and those neighbors. Set the shortest distance of the root to zero.
2. **Iteration:** Repeat the following two steps until all nodes are added to the path:
   a. **Adding the next node to the path:** Search the nodes not in the *path*. Select the one with minimum shortest distance and add it to the *path*.
   b. **Updating:** Update the shortest distance for all remaining nodes using the shortest distance of the node just moved to the path in step 2.

$$D_j = \text{minimum } (D_j, D_i + c_{ij}) \quad \text{for all remaining nodes}$$

Table 11.3 shows the simple version of this algorithm.

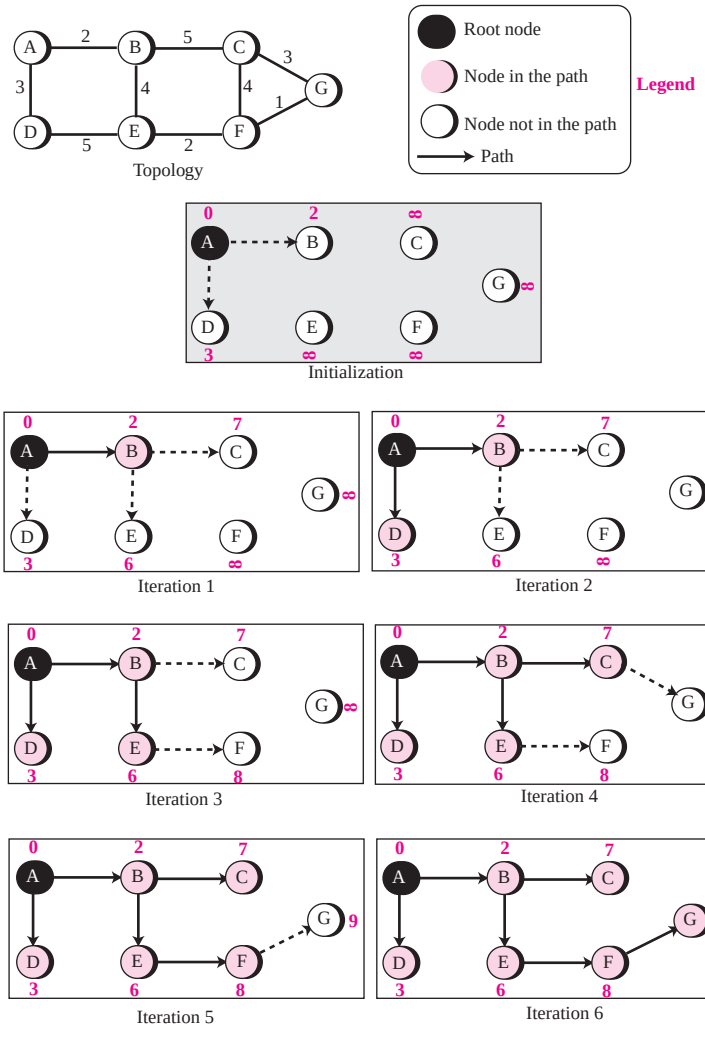
**Table 11.3** *Dijkstra's Algorithm*

```
Dijkstra ( )
{
   // Initialization
   Path = {s}          // s means self
   for (i = 1 to N)
   {
      if(i is a neighbor of s and i ≠ s)   D_i = c_si
      if (i is not a neighbor of s)        D_i = ∞
   }
   D_s = 0

} // Dijkstra
   // Iteration
   Repeat
   {
       // Finding the next node to be added
    Path = Path ∪ i  if D_i is minimum among all remaining nodes

       // Update the shortest distance for the rest
    for (j = 1 to M)    // M number of remaining nodes
    {
         D_j = minimum (D_j ,  D_j  + c_ij)
     }
   } until (all nodes included in the path, M = 0)
```

Figure 11.19 shows the formation of the shortest path tree for the graph of seven nodes. All the nodes in the graph have the same topology, but each node creates a different shortest path tree with itself as the root of the tree.We show the tree created by node A. We need to go through an initialization step and six iterations to find the shortest tree.

In the initialization step, node A selects itself as the root. It then assigns shortest path distances to each node on the topology. The nodes that are not neighbors of A receive a shortest path distance value of infinity.

**Figure 11.19** *Forming shortest path three for router A in a graph*

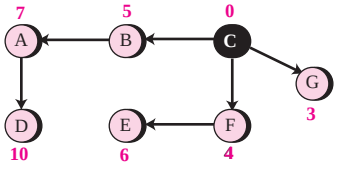



In each iteration, the next node with minimum distance is selected and added to the path. Then all shortest distances are updated with respect to the last node selected. For example, in the first iteration, node B is selected and added to the path and the shortest distances are updated with respect to node B (The shortest distances for C and E are changed, but for the others remain the same). After six iterations, the shortest path tree is found for node A. Note that in iteration 4, the shortest path to G is found via C, but in iteration 5, a new shortest route is discovered (via G); the previous path is erased and the new one is added.

### Example 11.6

To show that the shortest path tree for each node is different, we found the shortest path tree as seen by node C (Figure 11.20). We leave the detail as an exercise.

**Figure 11.20** *Example 11.6*




### *Calculation of Routing Table from Shortest Path Tree*

Each node uses the shortest path tree found in the previous discussion to construct its routing table. The routing table shows the cost of reaching each node from the root. Table 11.4 shows the routing table for node A using the shortest path tree found in Figure 11.19.

**Table 11.4** *Routing Table for Node A*

| *Destination* | *Cost* | *Next Router* |
|---|---|---|
| A | 0 | — |
| B | 2 | — |
| C | 7 | B |
| D | 3 | — |
| E | 6 | B |
| F | 8 | B |
| G | 9 | B |

## 11.6 OSPF

The **Open Shortest Path First (OSPF) protocol** is an intradomain routing protocol based on link state routing. Its domain is also an autonomous system.

### Areas

To handle routing efficiently and in a timely manner, OSPF divides an autonomous system into areas. An **area** is a collection of networks, hosts, and routers all contained within an autonomous system. An autonomous system can be divided into many different areas. All networks inside an area must be connected.

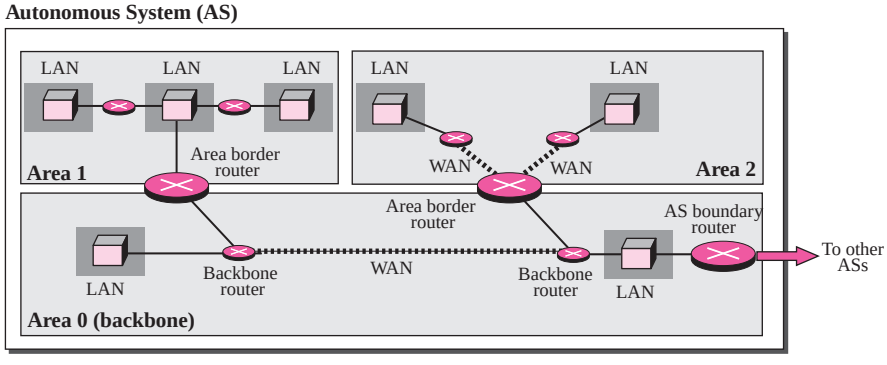
Routers inside an area flood the area with routing information. At the border of an area, special routers called **area border routers** summarize the information about the area and send it to other areas. Among the areas inside an autonomous system is a special area called the *backbone;* all of the areas inside an autonomous system must be

connected to the backbone. In other words, the backbone serves as a primary area and the other areas as secondary areas. This does not mean that the routers within areas cannot be connected to each other, however. The routers inside the backbone are called the ***backbone routers.*** Note that a backbone router can also be an area border router.

If, because of some problem, the connectivity between a backbone and an area is broken, a **virtual link** between routers must be created by the administration to allow continuity of the functions of the backbone as the primary area.

Each area has an area identification. The area identification of the backbone is zero. Figure 11.21 shows an autonomous system and its areas.

**Figure 11.21** *Areas in an autonomous system*



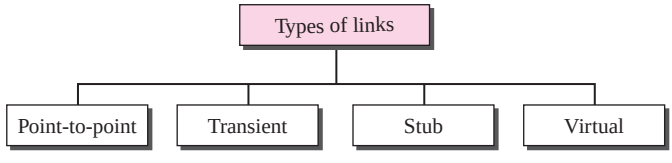

## Metric

The OSPF protocol allows the administrator to assign a cost, called the **metric,** to each route. The metric can be based on a type of service (minimum delay, maximum throughput, and so on). As a matter of fact, a router can have multiple routing tables, each based on a different type of service.

## Types of Links

In OSPF terminology, a connection is called a *link*. Four types of links have been defined: point-to-point, transient, stub, and virtual (see Figure 11.22).

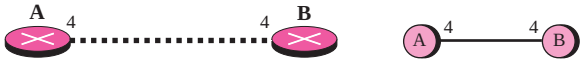
**Figure 11.22** *Types of links*



### Point-to-Point Link

A **point-to-point link** connects two routers without any other host or router in between. In other words, the purpose of the link (network) is just to connect the two routers. An example of this type of link is two routers connected by a telephone line or a T-line. There is no need to assign a network address to this type of link. Graphically, the routers are represented by nodes, and the link is represented by a bidirectional edge connecting the nodes. The metrics, which are usually the same, are shown at the two ends, one for each direction. In other words, each router has only one neighbor at the other side of the link (see Figure 11.23).

**Figure 11.23** *Point-to-point link*

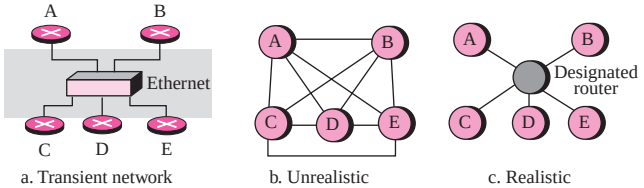



a. Point-to-point network b. Representation

### Transient Link

A **transient link** is a network with several routers attached to it. The data can enter through any of the routers and leave through any router. All LANs and some WANs with two or more routers are of this type. In this case, each router has many neighbors. For example, consider the Ethernet in Figure 11.24a. Router A has routers B, C, D, and E as neighbors. Router B has routers A, C, D, and E as neighbors. If we want to show the neighborhood relationship in this situation, we have the graph shown in Figure 11.24b.

**Figure 11.24** *Transient link*




a. Transient network b. Unrealistic c. Realistic

This is neither efficient nor realistic. It is not efficient because each router needs to advertise the neighborhood to four other routers, for a total of 20 advertisements. It is not realistic, because there is no single network (link) between each pair of routers; there is only one network that serves as a crossroad between all five routers.

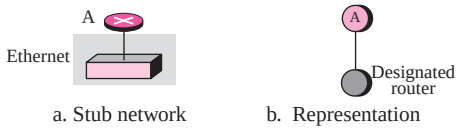
To show that each router is connected to every other router through one single network, the network itself is represented by a node. However, because a network is not a machine, it cannot function as a router. One of the routers in the network takes this responsibility. It is assigned a dual purpose; it is a true router and a designated router. We can use the topology shown in Figure 11.24c to show the connections of a transient network.

Now each router has only one neighbor, the designated router (network). On the other hand, the designated router (the network) has five neighbors. We see that the number of neighbor announcements is reduced from 20 to 10. Still, the link is represented as a bidirectional edge between the nodes. However, while there is a metric from each node to the designated router, there is no metric from the designated router to any other node. The reason is that the designated router represents the network. We can only assign a cost to a packet that is passing through the network. We cannot charge for this twice. When a packet enters a network, we assign a cost; when a packet leaves the network to go to the router, there is no charge.

### *Stub Link*

A **stub link** is a network that is connected to only one router. The data packets enter the network through this single router and leave the network through this same router. This is a special case of the transient network. We can show this situation using the router as a node and using the designated router for the network. However, the link is only one-directional, from the router to the network (see Figure 11.25).

**Figure 11.25** *Stub link*




a. Stub network b. Representation

### *Virtual Link*

When the link between two routers is broken, the administration may create a **virtual link** between them using a longer path that probably goes through several routers.

## Graphical Representation

Let us now examine how an AS can be represented graphically. Figure 11.26 shows a small AS with seven networks and six routers. Two of the networks are point-to-point networks. We use symbols such as N1 and N2 for transient and stub networks. There is no need to assign an identity to a point-to-point network. The figure also shows the graphical representation of the AS as seen by OSPF.

We have used color nodes for the routers and shaded nodes for the networks (represented by designated routers). However, OSPF sees both as nodes. Note that we have three stub networks.

## OSPF Packets

OSPF uses five different types of packets: *hello, database description, link state request, link state update,* and *link state acknowledgment* (see Figure 11.27). The most important one is the link state update that itself has five different kinds.

**Figure 11.26** *Example of an AS and its graphical representation in OSPF*

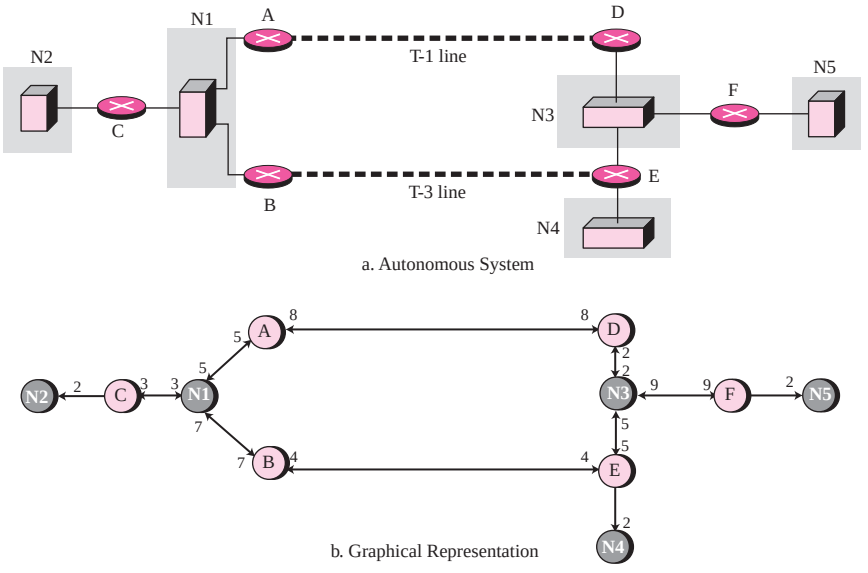
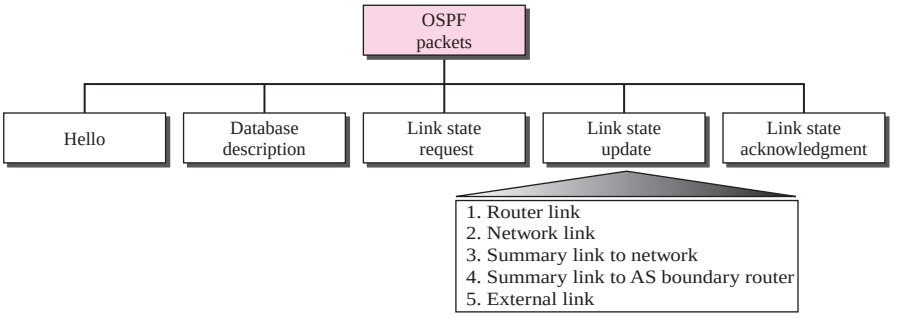



**Figure 11.27** *Types of OSPF packets*

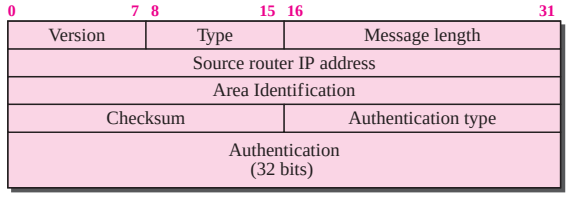



### *Common Header*

All OSPF packets have the same common header (see Figure 11.28). Before studying the different types of packets, let us talk about this common header.

- **Version.** This 8-bit field defines the version of the OSPF protocol. It is currently version 2.
- **Type.** This 8-bit field defines the type of the packet. As we said before, we have five types, with values 1 to 5 defining the types.
- **Message length.** This 16-bit field defines the length of the total message including the header.

**Figure 11.28** *OSPF common header*

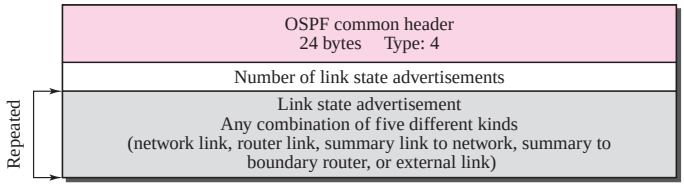



- **Source router IP address.** This 32-bit field defines the IP address of the router that sends the packet.
- **Area identification.** This 32-bit field defines the area within which the routing takes place.
- **Checksum.** This field is used for error detection on the entire packet excluding the authentication type and authentication data field.
- **Authentication type.** This 16-bit field defines the authentication protocol used in this area. At this time, two types of authentication are defined: 0 for none and 1 for password.
- **Authentication.** This 64-bit field is the actual value of the authentication data. In the future, when more authentication types are defined, this field will contain the result of the authentication calculation. For now, if the authentication type is 0, this field is filled with 0s. If the type is 1, this field carries an eight-character password.

## Link State Update Packet

We first discuss the **link state update packet,** the heart of the OSPF operation. It is used by a router to advertise the states of its links. The general format of the link state update packet is shown in Figure 11.29.

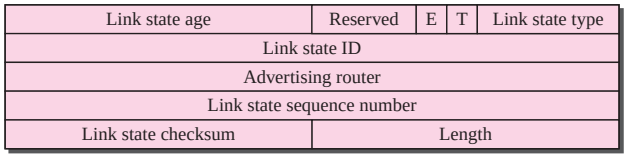
**Figure 11.29** *Link state update packet*




Each update packet may contain several different LSAs. All five kinds have the same general header. This general header is shown in Figure 11.30 and described below:

- **Link state age.** This field indicates the number of seconds elapsed since this message was first generated. Recall that this type of message goes from router to router (flooding). When a router creates the message, the value of this field is 0. When

**Figure 11.30** *LSA general header*




each successive router forwards this message, it estimates the transit time and adds it to the cumulative value of this field.

- **E flag.** If this 1-bit flag is set to 1, it means that the area is a stub area. A stub area is an area that is connected to the backbone area by only one path.
- **T flag.** If this 1-bit flag is set to 1, it means that the router can handle multiple types of service.
- **Link state type.** This field defines the LSA type. As we discussed before, there are five different advertisement types: router link (1), network link (2), summary link to network (3), summary link to AS boundary router (4), and external link (5).
- **Link state ID.** The value of this field depends on the type of link. For type 1 (router link), it is the IP address of the router. For type 2 (network link), it is the IP address of the designated router. For type 3 (summary link to network), it is the address of the network. For type 4 (summary link to AS boundary router), it is the IP address of the AS boundary router. For type 5 (external link), it is the address of the external network.
- **Advertising router.** This is the IP address of the router advertising this message.
- **Link state sequence number.** This is a sequence number assigned to each link state update message.
- **Link state checksum.** This is not the usual checksum. Instead, the value of this field is calculated using *Fletcher's checksum* (see Appendix C), which is based on the whole packet except for the age field.
- **Length.** This defines the length of the whole packet in bytes.

### *Router Link LSA*

A router link defines the links of a true router. A true router uses this advertisement to announce information about all of its links and what is at the other side of the link (neighbors). See Figure 11.31 for a depiction of a router link.

The router link LSA advertises all of the links of a router (true router). The format of the router link packet is shown in Figure 11.32.

The fields of the router link LSA are as follows:

- **Link ID.** The value of this field depends on the type of link. Table 11.5 shows the different link identifications based on link type.
- **Link data.** This field gives additional information about the link. Again, the value depends on the type of the link (see Table 11.5).

**Figure 11.31** *Router link*

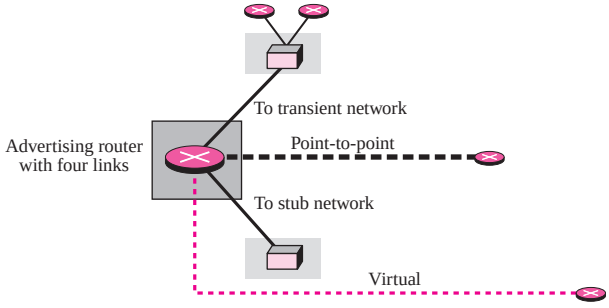



**Figure 11.32** *Router link LSA*

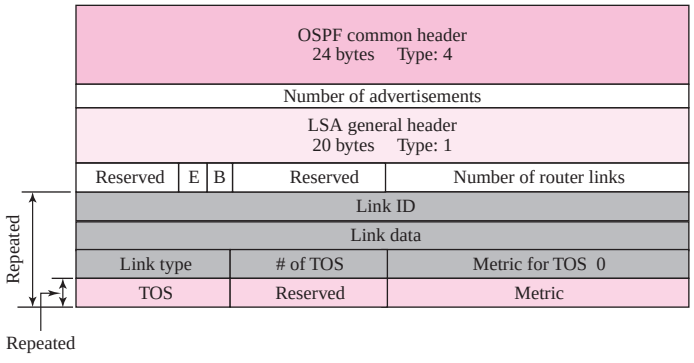



**Table 11.5** *Link Types, Link Identification, and Link Data*

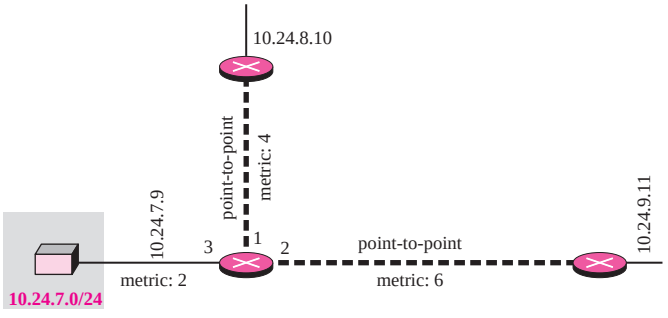
| *Link Type* | *Link Identification* | *Link Data* |
|---|---|---|
| Type 1: Point-to-point | Address of neighbor router | Interface number |
| Type 2: Transient | Address of designated router | Router address |
| Type 3: Stub | Network address | Network mask |
| Type 4: Virtual | Address of neighbor router | Router address |

- **Link type.** Four different types of links are defined based on the type of network to which the router is connected (see Table 11.5).
- **Number of types of service (TOS).** This field defines the number of types of services announced for each link.
- **Metric for TOS 0.** This field defines the metric for the default type of service (TOS 0).
- **TOS.** This field defines the type of service.
- **Metric.** This field defines the metric for the corresponding TOS.

### Example 11.7

Give the router link LSA sent by router 10.24.7.9 in Figure 11.33.

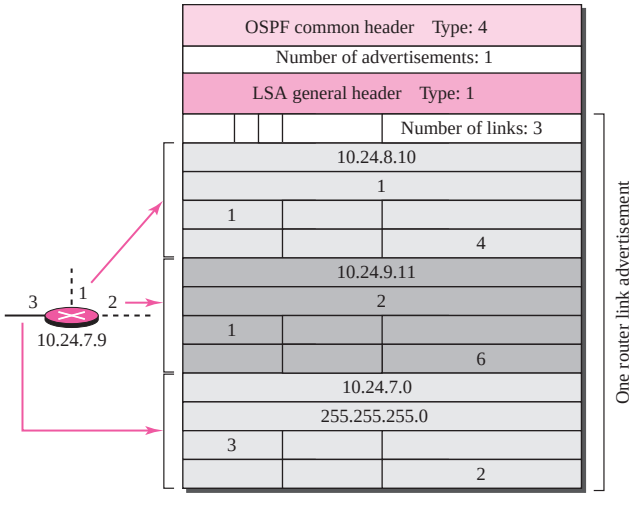
**Figure 11.33** *Example 11.7*




**Solution**

This router has three links: two of type 1 (point-to-point) and one of type 3 (stub network). Figure 11.34 shows the router link LSA.

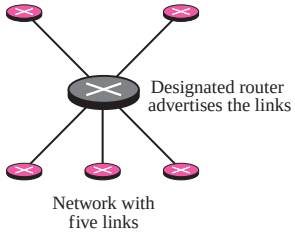
**Figure 11.34** *Solution to Example 11.8*




### *Network Link LSA*

A network link defines the links of a network. A designated router, on behalf of the transient network, distributes this type of LSP packet. The packet announces the existence of all of the routers connected to the network (see Figure 11.35). The format of the

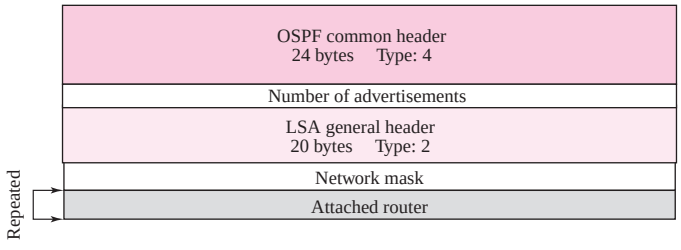
**Figure 11.35** *Network link*




network link advertisement is shown in Figure 11.36. The fields of the network link LSA are as follows:

- **Network mask.** This field defines the network mask.
- **Attached router.** This repeated field defines the IP addresses of all attached routers.

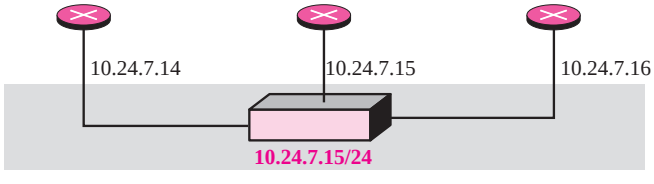
**Figure 11.36** *Network link advertisement format*




### Example 11.8

Give the network link LSA in Figure 11.37.

**Figure 11.37** *Example 4*




**Solution**

The network for which the network link advertises has three routers attached. The LSA shows the mask and the router addresses. Figure 11.38 shows the network link LSA.

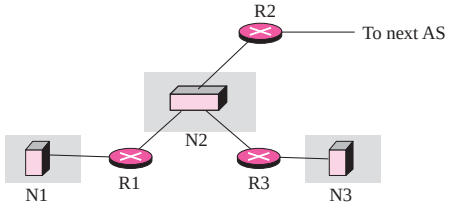
**Figure 11.38** *Solution to Example 4*

| OSPF common header Type: 4 |
|---|
| Number of advertisements: 1 |
| LSA general header Type: 2 |
| 255.255.255.0 |
| 10.24.7.14 |
| 10.24.7.15 |
| 10.24.7.16 |

### Example 11.9

In Figure 11.39, which router(s) sends out router link LSAs?

**Figure 11.39** *Example 11.9 and Example 11.10*




**Solution**

All routers advertise router link LSAs.

- **a.** R1 has two links, N1 and N2.
- **b.** R2 has one link, N1.
- **c.** R3 has two links, N2 and N3.

### Example 11.10

In Figure 11.39, which router(s) sends out the network link LSAs?

**Solution**

All three networks must advertise network links:

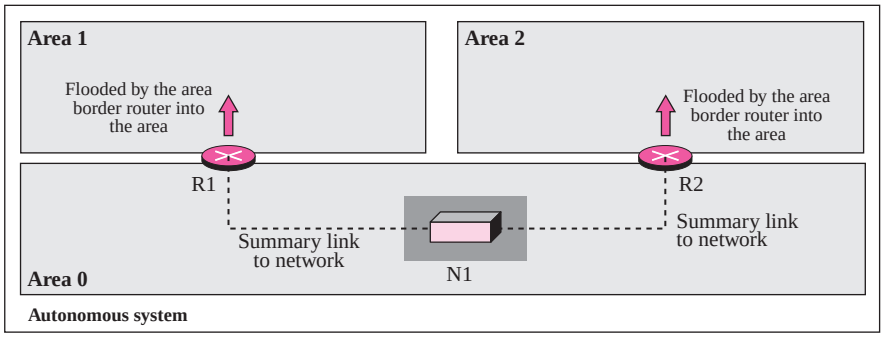
- **a.** Advertisement for N1 is done by R1 because it is the only attached router and therefore the designated router.
- **b.** Advertisement for N2 can be done by either R1, R2, or R3, depending on which one is chosen as the designated router.
- **c.** Advertisement for N3 is done by R3 because it is the only attached router and therefore the designated router.

### *Summary Link to Network LSA*

Router link and network link advertisements flood the area with information about the router links and network links inside an area. But a router must also know about the networks outside its area; the area border routers can provide this information. An area border router is active in more than one area. It receives router link and network link

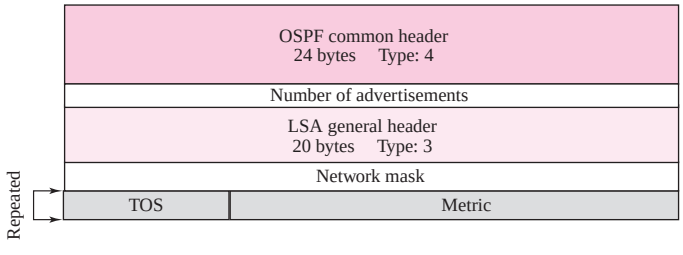
advertisements, and, as we will see, creates a routing table for each area. For example, in Figure 11.40, router R1 is an area border router. It has two routing tables, one for area 1 and one for area 0. R1 floods area 1 with information about how to reach a network located in area 0. In the same way, router R2 floods area 2 with information about how to reach the same network in area 0.

**Figure 11.40** *Summary link to network*




The summary link to network LSA is used by the area border router to announce the existence of other networks outside the area. The summary link to network advertisement is very simple. It consists of the network mask and the metric for each type of service. Note that each advertisement announces only one single network. If there is more than one network, a separate advertisement must be issued for each. The reader may ask why only the mask of the network is advertised. What about the network address itself? The IP address of the advertising router is announced in the header of the link state advertisement. From this information and the mask, one can deduce the network address. The format of this advertisement is shown in Figure 11.41. The fields of the summary link to network LSA are as follows:

**Figure 11.41** *Summary link to network LSA*




- **Network mask.** This field defines the network mask.
- **TOS.** This field defines the type of service.
- **Metric.** This field defines the metric for the type of service defined in the TOS field.

### *Summary Link to AS Boundary Router LSA*

The previous advertisement lets every router know the cost to reach all of the networks inside the autonomous system. But what about a network outside the autonomous system? If a router inside an area wants to send a packet outside the autonomous system, it should first know the route to an autonomous boundary router; the summary link to AS boundary router provides this information. The area border routers flood their areas with this information (see Figure 11.42). This packet is used to announce the route to an AS boundary router. Its format is the same as the previous summary link. The packet just defines the network to which the AS boundary router is attached. If a message can reach the network, it can be picked up by the AS boundary router. The format of the packet is shown in Figure 11.43. The fields are the same as the fields in the summary link to network advertisement message.

**Figure 11.42** *Summary link to AS boundary router*

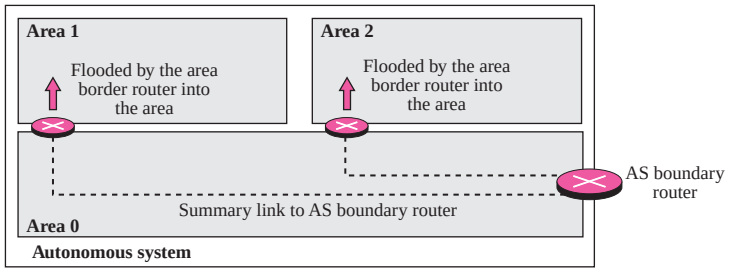
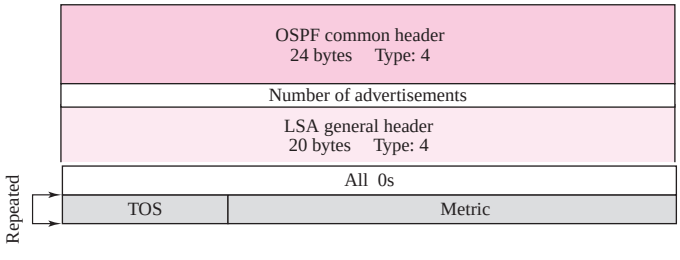



**Figure 11.43** *Summary link to AS boundary router LSA*




### *External Link LSA*

Although the previous advertisement lets each router know the route to an AS boundary router, this information is not enough. A router inside an autonomous system wants to know which networks are available outside the autonomous system; the external link advertisement provides this information. The AS boundary router floods the autonomous system with the cost of each network outside the autonomous system

using a routing table created by an interdomain routing protocol. Each advertisement announces one single network. If there is more than one network, separate announcements are made. Figure 11.44 depicts an external link. This is used to announce all the networks outside the AS. The format of the LSA is similar to the summary link to the AS boundary router LSA, with the addition of two fields. The AS boundary router may define a forwarding router that can provide a better route to the destination. The packet also can include an external route tag, used by other protocols, but not by OSPF. The format of the packet is shown in Figure 11.45.

**Figure 11.44** *External link*

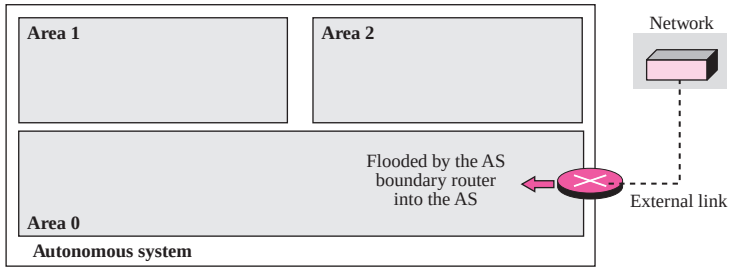
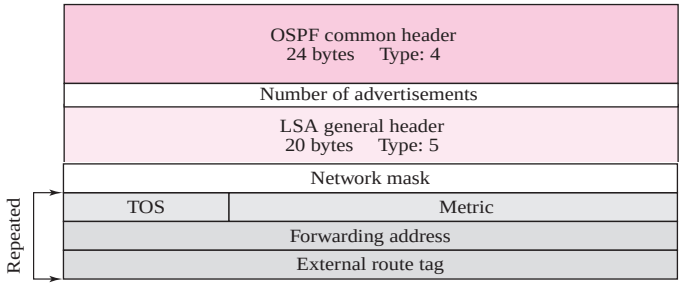



**Figure 11.45** *External link LSA*



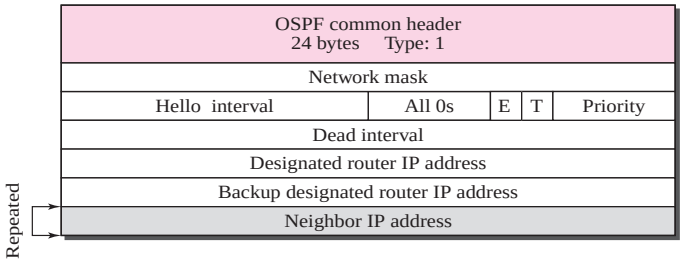

## Other Packets

Now we discuss four other packet types (See Figure 11.27). They are not used as LSAs, but are essential to the operation of OSPF.

### *Hello Message*

OSPF uses the **hello message** to create neighborhood relationships and to test the reachability of neighbors. This is the first step in link state routing. Before a router can flood all of the other routers with information about its neighbors, it must first greet its neighbors. It must know if they are alive, and it must know if they are reachable (see Figure 11.46).

**Figure 11.46** *Hello packet*



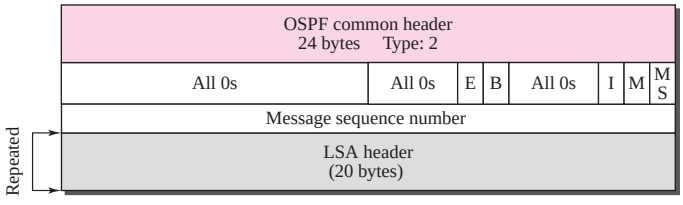

- **Network mask.** This 32-bit field defines the network mask of the network over which the hello message is sent.
- **Hello interval.** This 16-bit field defines the number of seconds between hello messages.
- **E flag.** This is a 1-bit flag. When it is set, it means that the area is a stub area.
- **T flag.** This is a 1-bit flag. When it is set, it means that the router supports multiple metrics.
- **Priority.** This field defines the priority of the router. The priority determines the selection of the designated router. After all neighbors declare their priorities, the router with the highest priority is chosen as the designated router. The one with the second highest priority is chosen as the backup designated router. If the value of this field is 0, it means that the router never wants to be a designated or a backup designated router.
- **Dead interval.** This 32-bit field defines the number of seconds that must pass before a router assumes that a neighbor is dead.
- **Designated router IP address.** This 32-bit field is the IP address of the designated router for the network over which the message is sent.
- **Backup designated router IP address.** This 32-bit field is the IP address of the backup designated router for the network over which the message is sent.
- **Neighbor IP address.** This is a repeated 32-bit field that defines the routers that have agreed to be the neighbors of the sending router. In other words, it is a current list of all the neighbors from which the sending router has received the hello message.

### *Database Description Message*

When a router is connected to the system for the first time or after a failure, it needs the complete link state database immediately. It cannot wait for all link state update packets to come from every other router before making its own database and calculating its routing table. Therefore, after a router is connected to the system, it sends hello packets to greet its neighbors. If this is the first time that the neighbors hear from the router, they send a database description message. The database description packet does not

contain complete database information; it only gives an outline, the title of each line in the database. The newly connected router examines the outline and finds out which lines of information it does not have. It then sends one or more link state request packets to get full information about that particular link. When two routers want to exchange database description packets, one of them takes the role of master and the other the role of slave. Because the message can be very long, the contents of the database can be divided into several messages. The format of the database description packet is shown in Figure 11.47. The fields are as follows:

**Figure 11.47** *Database description packet*

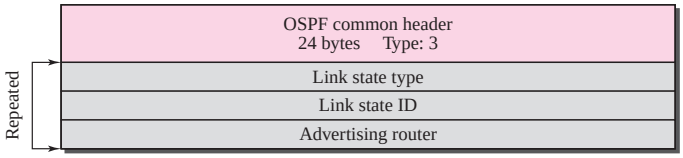



- **E flag.** This 1-bit flag is set to 1 if the advertising router is an autonomous boundary router (*E* stands for external).
- **B flag.** This 1-bit flag is set to 1 if the advertising router is an area border router.
- **I flag.** This 1-bit field, the *initialization* flag, is set to 1 if the message is the first message.
- **M flag.** This 1-bit field, the *more* flag, is set to 1 if this is not the last message.
- **M/S flag.** This 1-bit field, the *master/slave* bit, indicates the origin of the packet: master (M/S = 1) or slave (M/S = 0).
- **Message sequence number.** This 32-bit field contains the sequence number of the message. It is used to match a request with the response.
- **LSA header.** This 20-byte field is used in each LSA. The format of this header is discussed in the link state update message section. This header gives the outline of each link, without details. It is repeated for each link in the link state database.

### *Link State Request Packet*

The format of the **link state request packet** is shown in Figure 11.48. This is a packet that is sent by a router that needs information about a specific route or routes. It is answered with a link state update packet. It can be used by a newly connected router to request more information about some routes after receiving the database description packet. The three fields here are part of the LSA header, which has already been discussed. Each set of the three fields is a request for one single LSA. The set is repeated if more than one advertisement is desired.

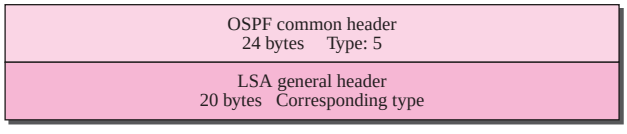
**Figure 11.48** *Link state request packet*




### *Link State Acknowledgment Packet*

OSPF makes routing more reliable by forcing every router to acknowledge the receipt of every link state update packet. The format of the **link state acknowledgment packet** is shown in Figure 11.49. It has the common OSPF header and the general LSA header. These two sections are sufficient to acknowledge a packet.

**Figure 11.49** *Link state acknowledgment packet*




## Encapsulation

OSPF packets are encapsulated in IP datagrams. They contain the acknowledgment mechanism for flow and error control. They do not need a transport layer protocol to provide these services.

**OSPF packets are encapsulated in IP datagrams.**

# 11.7 PATH VECTOR ROUTING

Distance vector and link state routing are both *interior* routing protocols. They can be used inside an autonomous system as intra-domain or intra-AS (as sometimes are called), but not between autonomous systems. Both of these routing protocols become intractable when the domain of operation becomes large. Distance vector routing is subject to instability if there is more than a few hops in the domain of operation. Link state routing needs a huge amount of resources to calculate routing tables. It also creates heavy traffic because of flooding. There is a need for a third routing protocol which we call **path vector routing.**

Path vector routing is exterior routing protocol proved to be useful for inter-domain or inter-AS routing as it is sometimes called. In distance vector routing, a

router has a list of networks that can be reached in the same AS with the corresponding cost (number of hops). In path vector routing, a router has a list of networks that can be reached with the path (list of ASs to pass) to reach each one. In other words, the domain of operation of the distance vector routing is a single AS; the domain of operation of the path vector routing is the whole Internet. The distance vector routing tells us the distance to each network; the path vector routing tells us the path.

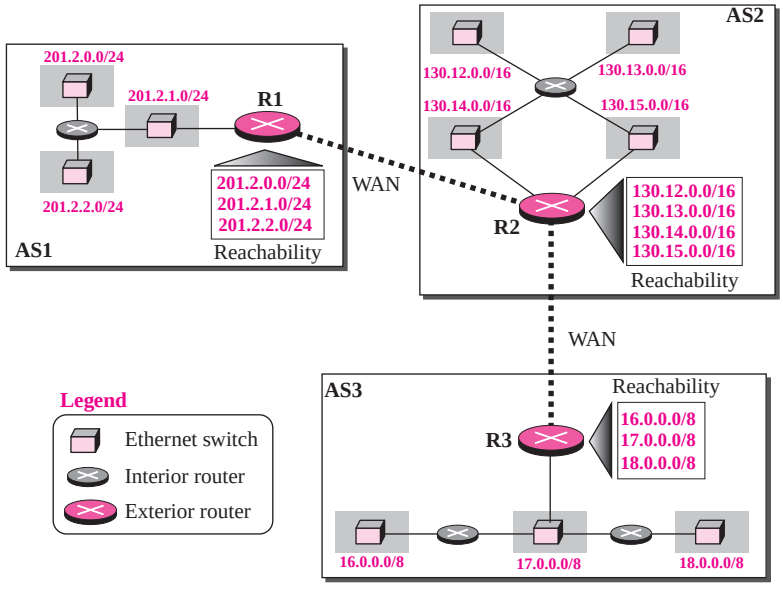
### Example 11.11

The difference between the distance vector routing and path vector routing can be compared to the difference between a national map and an international map. A national map can tell us the road to each city and the distance to be travelled if we choose a particular route; an international map can tell us which cities exist in each country and which countries should be passed before reaching that city.

## Reachability

To be able to provide information to other ASs, each AS must have at least one path vector routing that collects *reachability* information about each network in that AS. The information collected in this case only means which network, identified by its network address (CIDR prefix), exists (can be reached in this AS). In other words, the AS needs to have a list of existing networks in its territory. Figure 11.50 shows three ASs. Each distance vector (exterior router) has created a list which shows which network is reachable in that AS.

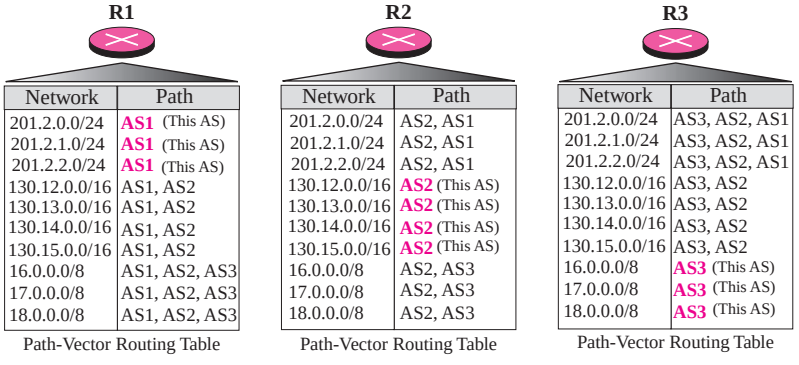
**Figure 11.50** *Reachability*



## Routing Tables

A path vector routing table for each router can be created if ASs share their reachability list with each other. In Figure 11.50, router R1 in AS1 can send its reachability list to router R2. Router R2, after combining its reachability list, can send the result to both R1 and R3. Router R3 can send its reachability list to R2, which in turn improves its routing table, and so on. Figure 11.51 shows the routing table for each router after all three routers have updated their routing table. Router R1 knows that if a packet arrives for the network 201.2.2.0/24, this network is in AS1 (at home), but if a packet arrives for the network 130.14.0.0/16, the packet should travel from AS1 to AS2 to reach its destination network. On the other hand, if router R2 receives a packet destined for the network 22.0.0.0.8, the router knows that it should travel from AS2 to AS3 to reach its destination. We can compare these routing tables with the distance vector routing table to see the differences.

**Figure 11.51** *Stabilized tables for three autonomous systems*




R1

| Network | Path |
|---|---|
| 201.2.0.0/24 | **AS1** (This AS) |
| 201.2.1.0/24 | **AS1** (This AS) |
| 201.2.2.0/24 | **AS1** (This AS) |
| 130.12.0.0/16 | AS1, AS2 |
| 130.13.0.0/16 | AS1, AS2 |
| 130.14.0.0/16 | AS1, AS2 |
| 130.15.0.0/16 | AS1, AS2 |
| 16.0.0.0/8 | AS1, AS2, AS3 |
| 17.0.0.0/8 | AS1, AS2, AS3 |
| 18.0.0.0/8 | AS1, AS2, AS3 |

Path-Vector Routing Table

R2

| Network | Path |
|---|---|
| 201.2.0.0/24 | AS2, AS1 |
| 201.2.1.0/24 | AS2, AS1 |
| 201.2.2.0/24 | AS2, AS1 |
| 130.12.0.0/16 | **AS2** (This AS) |
| 130.13.0.0/16 | **AS2** (This AS) |
| 130.14.0.0/16 | **AS2** (This AS) |
| 130.15.0.0/16 | **AS2** (This AS) |
| 16.0.0.0/8 | AS2, AS3 |
| 17.0.0.0/8 | AS2, AS3 |
| 18.0.0.0/8 | AS2, AS3 |

Path-Vector Routing Table

R3

| Network | Path |
|---|---|
| 201.2.0.0/24 | AS3, AS2, AS1 |
| 201.2.1.0/24 | AS3, AS2, AS1 |
| 201.2.2.0/24 | AS3, AS2, AS1 |
| 130.12.0.0/16 | AS3, AS2 |
| 130.13.0.0/16 | AS3, AS2 |
| 130.14.0.0/16 | AS3, AS2 |
| 130.15.0.0/16 | AS3, AS2 |
| 16.0.0.0/8 | **AS3** (This AS) |
| 17.0.0.0/8 | **AS3** (This AS) |
| 18.0.0.0/8 | **AS3** (This AS) |

Path-Vector Routing Table

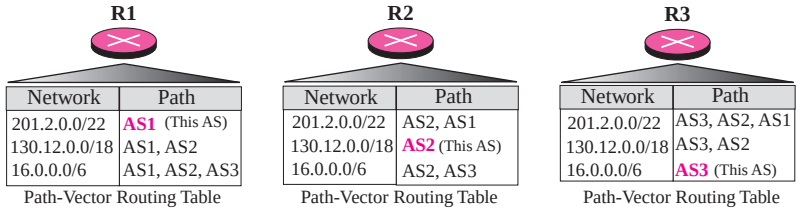
### *Loop Prevention*

The instability of distance vector routing and the creation of loops can be avoided in path vector routing. When a router receives a reachability information, it checks to see if its autonomous system is in the path list to any destination. If it is, looping is involved and that network-path pair is discarded.

### *Aggregation*

The path vector routing protocols normally support CIDR notation and the aggregation of addresses (if possible). This helps to make the path vector routing table simpler and exchange between routers faster. For example, the path vector routing table of Figure 11.51 can be aggregated to create shorter routing tables (Figure 11.52). Note that a range may also include a block that may not be in the corresponding AS. For example, the range 201.2.0.0/22 also includes the range 201.2.0.3/24, which is not the network address of any network in AS1. However, if this network exists in some other ASs, it eventually becomes part of the routing table. Based on the longest prefix principle we discussed in Chapter 6, this network address is above 201.2.0.0/22 and is searched first, which results in correct routing.

**Figure 11.52** *Routing table after aggregation*




### *Policy Routing*

**Policy routing** can be easily implemented through path vector routing. When a router receives a message, it can check the path. If one of the autonomous systems listed in the path is against its policy, it can ignore that path and that destination. It does not update its routing table with this path, and it does not send this message to its neighbors.

## 11.8 BGP

Border Gateway Protocol (BGP) is an interdomain routing protocol using path vector routing. It first appeared in 1989 and has gone through four versions.

### Types of Autonomous Systems

As we said before, the Internet is divided into hierarchical domains called autonomous systems (ASs). For example, a large corporation that manages its own network and has full control over it is an autonomous system. A local ISP that provides services to local customers is an autonomous system. Note that a single organization may choose to have multiple ASs because of geographical spread, different providers (ISPs), or even some local obstacles.

We can divide autonomous systems into three categories: stub, multihomed, and transit.

#### *Stub AS*

A stub AS has only one connection to another AS. The interdomain data traffic in a stub AS can be either created or terminated in the AS. The hosts in the AS can send data traffic to other ASs. The hosts in the AS can receive data coming from hosts in other ASs. Data traffic, however, cannot pass through a stub AS. A stub AS is either a source or a sink. A good example of a stub AS is a small corporation or a small local ISP.

#### *Multihomed AS*

A multihomed AS has more than one connection to other ASs, but it is still only a source or sink for data traffic. It can receive data traffic from more than one AS. It can send data traffic to more than one AS, but there is no transient traffic. It does not allow data coming from one AS and going to another AS to pass through. A good example of a multihomed AS is a large corporation that is connected to more than one regional or national AS that does not allow transient traffic.

### *Transit AS*

A transit AS is a multihomed AS that also allows transient traffic. Good examples of transit ASs are national and international ISPs (Internet backbones).

### *CIDR*

BGP uses classless interdomain routing addresses. In other words, BGP uses a prefix, as discussed in Chapter 5, to define a destination address. The address and the number of bits (prefix length) are used in updating messages.

## Path Attributes

In our previous example, we discussed a path for a destination network. The path was presented as a list of autonomous systems, but is, in fact, a list of attributes. Each attribute gives some information about the path. The list of attributes helps the receiving router make a better decision when applying its policy.

Attributes are divided into two broad categories: well-known and optional. A **well-known attribute** is one that every BGP router must recognize. An **optional attribute** is one that needs not be recognized by every router.

Well-known attributes are themselves divided into two categories: mandatory and discretionary. A *well-known mandatory attribute* is one that must appear in the description of a route. A *well-known discretionary attribute* is one that must be recognized by each router, but is not required to be included in every update message. One well-known mandatory attribute is ORIGIN. This defines the source of the routing information (RIP, OSPF, and so on). Another well-known mandatory attribute is AS_PATH. This defines the list of autonomous systems through which the destination can be reached. Still another well-known mandatory attribute is NEXT-HOP, which defines the next router to which the data packet should be sent.

The optional attributes can also be subdivided into two categories: transitive and nontransitive. An *optional transitive attribute* is one that must be passed to the next router by the router that has not implemented this attribute. An *optional nontransitive attribute* is one that must be discarded if the receiving router has not implemented it.

## BGP Sessions

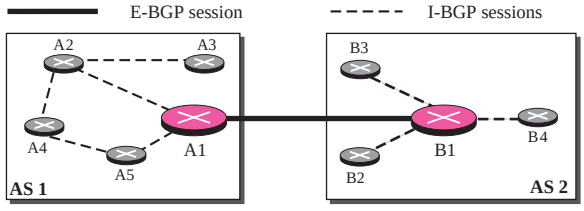
The exchange of routing information between two routers using BGP takes place in a session. A session is a connection that is established between two BGP routers only for the sake of exchanging routing information. To create a reliable environment, BGP uses the services of TCP. In other words, a session at the BGP level, as an application program, is a connection at the TCP level. However, there is a subtle difference between a connection in TCP made for BGP and other application programs. When a TCP connection is created for BGP, it can last for a long time, until something unusual happens. For this reason, BGP sessions are sometimes referred to as *semipermanent connections*.

## External and Internal BGP

If we want to be precise, BGP can have two types of sessions: external BGP (E-BGP) and internal BGP (I-BGP) sessions. The E-BGP session is used to exchange information

between two speaker nodes belonging to two different autonomous systems. The I-BGP session, on the other hand, is used to exchange routing information between two routers inside an autonomous system. Figure 11.53 shows the idea.

**Figure 11.53** *Internal and external BGP sessions*

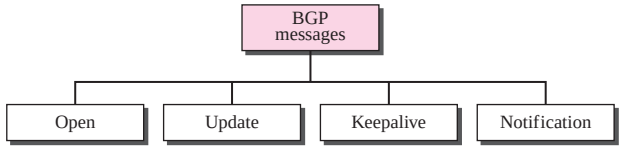



The session established between AS1 and AS2 is an E-BGP session. The two speaker routers exchange information they know about networks in the Internet. However, these two routers need to collect information from other routers in the autonomous systems. This is done using I-BGP sessions.

## Types of Packets

BGP uses four different types of messages: **open, update, keepalive,** and **notification** (see Figure 11.54).

**Figure 11.54** *Types of BGP messages*

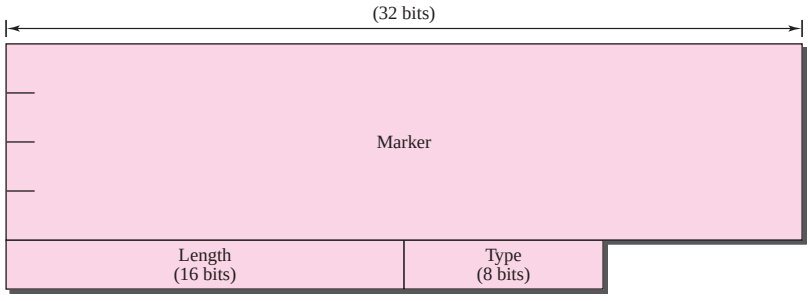



## Packet Format

All BGP packets share the same common header. Before studying the different types of packets, let us talk about this common header (see Figure 11.55). The fields of this header are as follows:

- **Marker.** The 16-byte marker field is reserved for authentication.
- **Length.** This 2-byte field defines the length of the total message including the header.
- **Type.** This 1-byte field defines the type of the packet. As we said before, we have four types, and the values 1 to 4 define those types.

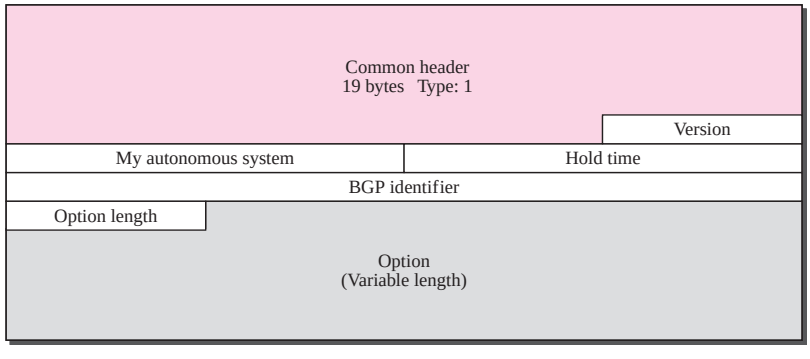
**Figure 11.55** *BGP packet header*




### *Open Message*

To create a neighborhood relationship, a router running BGP opens a TCP connection with a neighbor and sends an **open message.** If the neighbor accepts the neighborhood relationship, it responds with a **keepalive message,** which means that a relationship has been established between the two routers. See Figure 11.56 for a depiction of the open message format.

**Figure 11.56** *Open message*




The fields of the open message are as follows:

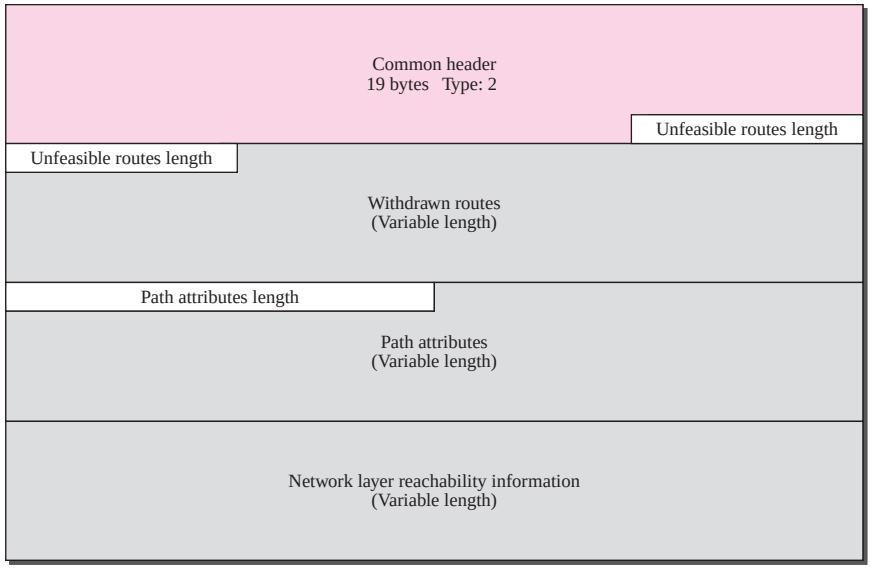
- **Version.** This 1-byte field defines the version of BGP. The current version is 4.
- **My autonomous system.** This 2-byte field defines the autonomous system number.
- **Hold time.** This 2-byte field defines the maximum number of seconds that can elapse until one of the parties receives a keepalive or update message from the other. If a router does not receive one of these messages during the hold time period, it considers the other party dead.

- **BGP identifier.** This 4-byte field defines the router that sends the open message. The router usually uses one of its IP addresses (because it is unique) for this purpose.
- **Option length.** The open message may contain some option parameters. In this case, this 1-byte field defines the length of the total option parameters. If there are no option parameters, the value of this field is zero.
- **Option parameters.** If the value of the option parameter length is not zero, it means that there are some option parameters. Each option parameter itself has two subfields: the length of the parameter and the parameter value. The only option parameter defined so far is authentication.

### *Update Message*

The update message is the heart of the BGP protocol. It is used by a router to withdraw destinations that have been advertised previously, announce a route to a new destination, or both. Note that BGP can withdraw several destinations that were advertised before, but it can only advertise one new destination in a single update message. The format of the update message is shown in Figure 11.57.

**Figure 11.57** *Update message*




The update message fields are listed below:

- **Unfeasible routes length.** This 2-byte field defines the length of the next field.
- **Withdrawn routes.** This field lists all the routes that must be deleted from the previously advertised list.

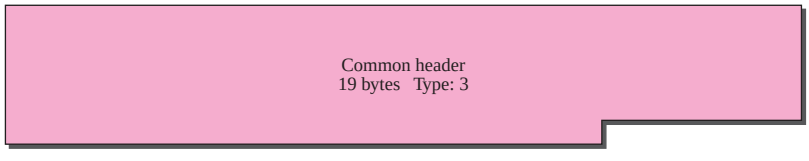
- **Path attributes length.** This 2-byte field defines the length of the next field.
- **Path attributes.** This field defines the attributes of the path (route) to the network whose reachability is being announced in this message.
- **Network layer reachability information (NLRI).** This field defines the network that is actually advertised by this message. It has a length field and an IP address prefix. The length defines the number of bits in the prefix. The prefix defines the common part of the network address. For example, if the network is 153.18.7.0/24, the length of the prefix is 24 and the prefix is 153.18.7. BGP4 supports classless addressing and CIDR.

**BGP supports classless addressing and CIDR.**

### *Keepalive Message*

The routers (called *peers* in BGP parlance) running the BGP protocols exchange keepalive messages regularly (before their hold time expires) to tell each other that they are alive. The keepalive message consists of only the common header shown in Figure 11.58.

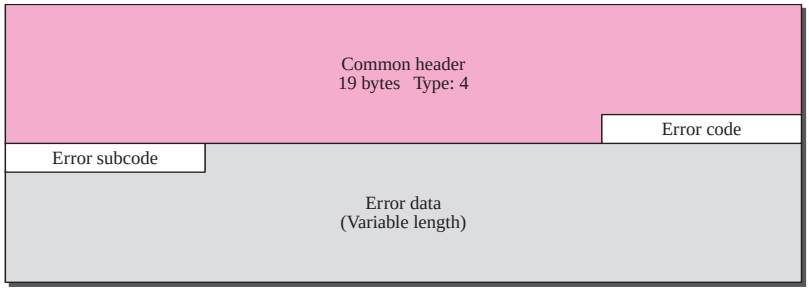
**Figure 11.58** *Keepalive message*




### *Notification Message*

A notification message is sent by a router whenever an error condition is detected or a router wants to close the connection. The format of the message is shown in Figure 11.59.The fields making up the notification message follow:

**Figure 11.59** *Notification message*



- **Error code.** This 1-byte field defines the category of the error. See Table 11.6.
- **Error subcode.** This 1-byte field further defines the type of error in each category.
- **Error data.** This field can be used to give more diagnostic information about the error.

**Table 11.6** *Error Codes*

| *Error Code* | *Error Code Description* | *Error Subcode Description* |
|---|---|---|
| 1 | Message header error | Three different subcodes are defined for this type of error: synchronization problem (1), bad message length (2), and bad message type (3). |
| 2 | Open message error | Six different subcodes are defined for this type of error: unsupported version number (1), bad peer AS (2), bad BGP identifier (3), unsupported optional parameter (4), authentication failure (5), and unacceptable hold time (6). |
| 3 | Update message error | Eleven different subcodes are defined for this type of error: malformed attribute list (1), unrecognized well-known attribute (2), missing well-known attribute (3), attribute flag error (4), attribute length error (5), invalid origin attribute (6), AS routing loop (7), invalid next hop attribute (8), optional attribute error (9), invalid network field (10), malformed AS_PATH (11). |
| 4 | Hold timer expired | No subcode defined. |
| 5 | Finite state machine error | This defines the procedural error. No subcode defined. |
| 6 | Cease | No subcode defined. |

### Encapsulation

BGP messages are encapsulated in TCP segments using the well-known port 179. This means that there is no need for error control and flow control. When a TCP connection is opened, the exchange of update, keepalive, and notification messages is continued until a notification message of type cease is sent.

**BGP uses the services of TCP on port 179.**

## 11.9 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give more information about unicast routing. In particular, we recommend [Com 06], [Pet & Dav 03], [Kur & Ros 08], [Per 00], [Gar & Vid 04], [Tan 03], [Sta 04], and [Moy 98].

### RFCs

Several RFCs are related to the protocols we discussed in this chapter. RIP is discussed in RFC1058 and RFC 2453. OSPF is discussed in RFC 1583 and RFC 2328. BGP is discussed in RFC 1654, RFC 1771, RFC 1773, RFC 1997, RFC 2439, RFC 2918, and RFC 3392.

## 11.10 KEY TERMS

area
area border router
autonomous system (AS)
autonomous system boundary router
backbone router
Bellman-Ford algorithm
Border Gateway Protocol (BGP)
cost
count to infinity
Dijkstra algorithm
distance vector routing
expiration timer
flooding
garbage collection timer
hello interval
hello message
hop count
inter-domain routing
intra-domain routing
keepalive message
link state request packet
link state routing
link state update packet
metric
notification message
open message
Open Shortest Path First (OSPF)
optional attribute
path vector routing
periodic timer
point-to-point link
poison reverse
Routing Information Protocol (RIP)
split horizon
stub link
transient link
update message
virtual link
well-known attribute

## 11.11 SUMMARY

- A metric is the cost assigned for passage of a packet through a network. A router consults its routing table to determine the best path for a packet.
- An autonomous system (AS) is a group of networks and routers under the authority of a single administration. RIP and OSPF are popular intradomain or intra-AS routing protocols (also called interior routing protocols) used to update routing tables in an AS. RIP is based on distance vector routing, in which each router shares, at regular intervals, its knowledge about the entire AS with its neighbors. OSPF divides an AS into areas, defined as collections of networks, hosts, and routers. OSPF is based on link state routing, in which each router sends the state of its neighborhood to every other router in the area.
- BGP is an interdomain or inter-AS routing protocol (also called exterior routing protocol) used to update routing tables. BGP is based on a routing protocol called path vector routing. In this protocol, the ASs through which a packet must pass are

explicitly listed. Path vector routing does not have the instability nor looping problems of distance vector routing. There are four types of BGP messages: open, update, keepalive, and notification.

## 11.12 PRACTICE SET

### Exercises

1. In RIP, why is the expiration timer value six times that of the periodic timer value?
2. How does the hop count limit alleviate RIP's problems?
3. Contrast and compare distance vector routing with link state routing.
4. Why do OSPF messages propagate faster than RIP messages?
5. What is the size of a RIP message that advertises only one network? What is the size of a RIP message that advertises *N* packets? Devise a formula that shows the relationship between the number of networks advertised and the size of a RIP message.
6. A router running RIP has a routing table with 20 entries. How many periodic timers are needed to handle this table? How many expiration timers are needed to handle this table? How many garbage collection timers are needed to handle this table if five routes are invalid?
7. A router using RIP has the routing table shown in Table 11.7.

**Table 11.7** *Routing Table for Exercise 7*

| *Destination* | *Cost* | *Next Router* |
|---|---|---|
| Net1 | 4 | B |
| Net2 | 2 | C |
| Net3 | 1 | F |
| Net4 | 5 | G |

   Show the RIP response message sent by this router.
8. The router in Exercise 7 receives a RIP message with four records from router C as shown below:

```
(Net1, 2), (Net2, 1), (Net3, 3), (Net4, 7)
```

   Show the updated routing table.
9. How many bytes are empty in a RIP message that advertises *N* networks?
10. Using Figure 11.26, do the following:
    a. Show the link state update/router link advertisement for router A.
    b. Show the link state update/router link advertisement for router D.
    c. Show the link state update/router link advertisement for router E.
    d. Show the link state update/network link advertisement for network N2.
    e. Show the link state update/network link advertisement for network N4.
    f. Show the link state update/network link advertisement for network N5.

**11.** In Figure 11.26,

- **a.** Assume that the designated router for network N1 is router A. Show the link state update/network link advertisement for this network.
- **b.** Assume that the designated router for network N3 is router D. Show the link state update/network link advertisement for this network.

**12.** Assign IP addresses to networks and routers in Figure 11.26. Now do the following:

- **a.** Show the OSPF hello message sent by router C.
- **b.** Show the OSPF database description message sent by router C.
- **c.** Show the OSPF link state request message sent by router C.

**13.** Show the autonomous system with the following specifications:

- **a.** There are eight networks (N1 to N8)
- **b.** There are eight routers (R1 to R8)
- **c.** N1, N2, N3, N4, N5, and N6 are Ethernet LANs
- **d.** N7 and N8 are point-to-point WANs
- **e.** R1 connects N1 and N2
- **f.** R2 connects N1 and N7
- **g.** R3 connects N2 and N8
- **h.** R4 connects N7 and N6
- **i.** R5 connects N6 and N3
- **j.** R6 connects N6 and N4
- **k.** R7 connects N6 and N5
- **l.** R8 connects N8 and N5

Now draw the graphical representation of the autonomous system of Exercise 29 as seen by OSPF. Which of the networks is a transient network? Which is a stub network?

**14.** In Figure 11.50,

- **a.** Show the BGP open message for router R1.
- **b.** Show the BGP update message for router R1.
- **c.** Show the BGP keepalive message for router R1.
- **d.** Show the BGP notification message for router R1.

**15.** In Figure 11.5, assume that the link between router A and router B fails (breaks). Show the changes in the routing table for routers in Figure 11.7.

**16.** In Figure 11.5, assume that the link between router C and router D fails (breaks). Show the changes in the routing table for routers in Figure 11.7.

**17.** Use the Bellman-Ford algorithm (Table 11.1) to find the shortest distance for all nodes in the graph of Figure 11.60.

**18.** Use the Dijkstra algorithm (Table 11.3) to find the shortest paths for all nodes in the graph of Figure 11.61.

**19.** Find the shortest path tree for node B in Figure 11.19.

**Figure 11.60** *Exercise 17*

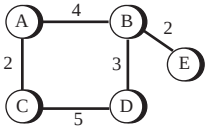
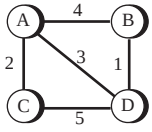



**Figure 11.61** *Exercise 18*




**20.** Find the shortest path tree for node E in Figure 11.19.

**21.** Find the shortest path tree for node G in Figure 11.19.

## Research Activities

**22.** Before BGP, there was a protocol called EGP. Find some information about this protocol. Find out why this protocol has not survived.

**23.** In UNIX, there are some programs under the general name *daemon*. Find the daemons that can handle routing protocols.

**24.** If you have access to a UNIX system, find some information about the *routed* program. How can this program help to trace the messages exchanged in RIP? Does *routed* support the other routing protocols we discussed in the chapter?

**25.** If you have access to a UNIX system, find some information about the *gated* program. Which routing protocols discussed in this chapter can be supported by *gated*?

**26.** There is a routing protocol called HELLO, which we did not discuss in this chapter. Find some information about this protocol.

CHAPTER

# 12

# *Multicasting and Multicast Routing Protocols*

In this chapter, we define multicasting and discuss multicast routing protocols. Multicast applications are in more and more demand everyday, but as we will see multicast routing is more difficult than unicast routing discussed in Chapter 11; a multicast router is response to send a copy of a multicast packet to all members of the corresponding group.

## OBJECTIVES

*The chapter has several objectives:*

- To compare and contrast unicasting, multicasting, and broadcasting communication.
- To define multicast addressing space in IPv4 and show the division of the space into several blocks.
- To discuss the IGMP protocol, which is responsible for collecting group membership information in a network.
- To discuss the general idea behind multicast routing protocols and their division into two categories based on the creation of the shortest path trees.
- To discuss *multicast link state routing* in general and its implementation in the Internet: a protocol named MOSPF.
- To discuss *multicast distance vector routing* in general and its implementation in the Internet: a protocol named DVMRP.
- To discuss core-based protocol (CBT) and briefly discuss two independent multicast protocols PIM-DM and PIM-SM.
- To discuss multicast backbone (MBONE) that shows how to create a tunnel when the multicast messages need to pass through an area with no multicast routers.

## 12.1 INTRODUCTION

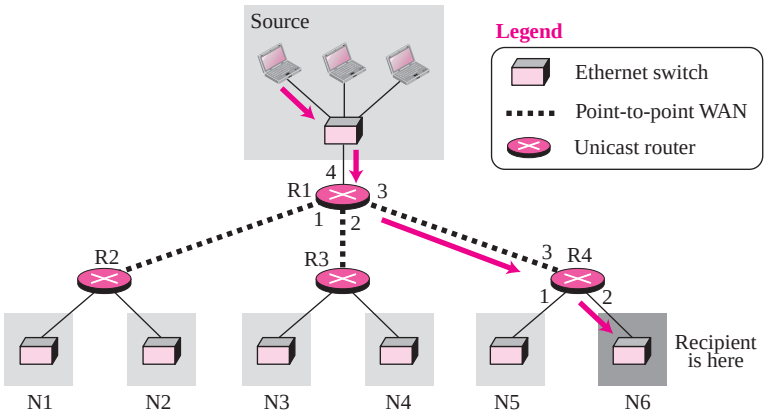
From the previous chapters in this part of the book, we have learned that forwarding a datagram by a router is normally based on the prefix of the destination address in the datagram, which defines the network to which the destination host is connected. Of course, address aggregation mechanism may combine several datagrams to be delivered to an ISP (which can be thought of as a large network holding some networks together) and then separate them to be delivered to their final destination networks, but the principle does not change. Aggregation just decreases the size of the prefix; separation just increases the size of the prefix.

Understanding the above forwarding principle, we can now define unicasting, multicasting, and broadcasting. Let us clarify these terms as they relate to the Internet.

### Unicasting

In *unicasting*, there is one source and one destination network. The relationship between the source and the destination network is one to one. Each router in the path of the datagram tries to forward the packet to one and only one of its interfaces. Figure 12.1 shows a small internet in which a unicast packet needs to be delivered from a source computer to a destination computer attached to N6. Router R1 is responsible to forward the packet only through interface 3; router R4 is responsible to forward the packet only through interface 2. When the packet arrives to N6, the delivery to the destination host is the responsibility of the network; it is either broadcast to all hosts or the smart Ethernet switch delivers it only to the destination host.

**Figure 12.1** *Unicasting*



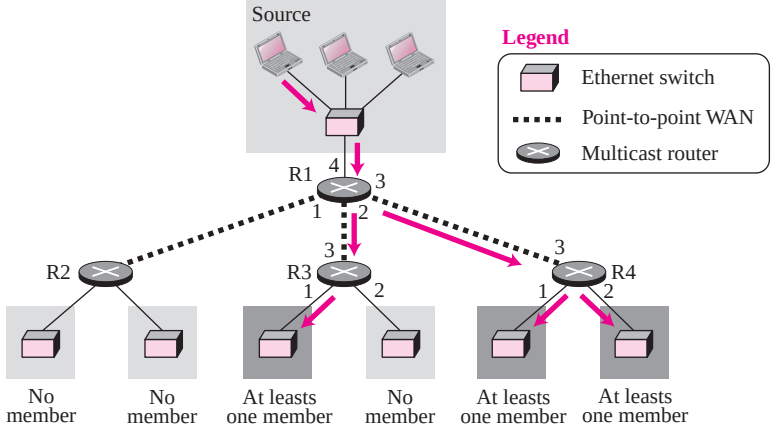
In unicasting, the routing table that defines the only output port for each datagram, is based on the optimum path as we saw in Chapter 11.

**In unicasting, the router forwards the received datagram through only one of its interfaces.**

## Multicasting

In *multicasting*, there is one source and a group of destinations. The relationship is one to many. In this type of communication, the source address is a unicast address, but the destination address is a group address, a group of one or more destination networks in which there is at least one member of the group that is interested in receiving the multicast datagram. The group address defines the members of the group. Figure 12.2 shows the same small internet in Figure 12.1, but the routers have been changed to multicast routers (or previous routers have been configured to do both types of job).

**Figure 12.2** *Multicasting*



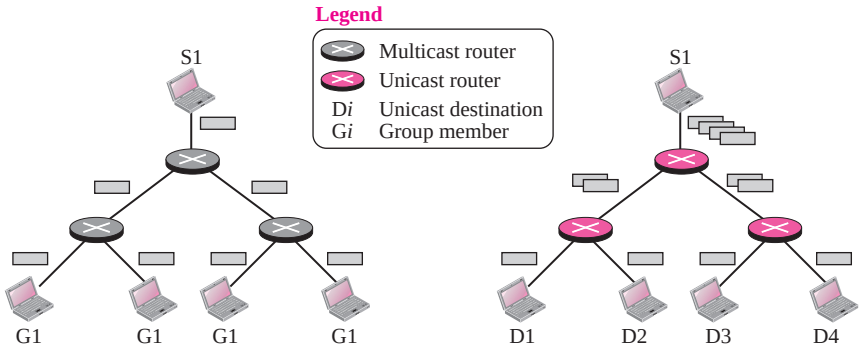

In multicasting, a multicast router may have to send out copies of the same datagram through more than one interface. In Figure 12.2, router R1 needs to send out the datagram through interfaces 2 and 3. Similarly, router R4 needs to send out the datagram through both its interfaces. Router R3, however, knows that there is no member belonging to this group in the area reached by interface 2; it only sends out the datagram through interface 1.

**In multicasting, the router may forward the received datagram through several of its interfaces.**

### *Multicasting versus Multiple Unicasting*

We need to distinguish between multicasting and multiple unicasting. Figure 12.3 illustrates both concepts.

**Figure 12.3** *Multicasting versus multiple unicasting*




a. Multicasting b. Multiple unicasting

Multicasting starts with one single packet from the source that is duplicated by the routers. The destination address in each packet is the same for all duplicates. Note that only one single copy of the packet travels between any two routers.

In *multiple unicasting*, several packets start from the source. If there are four destinations, for example, the source sends four packets, each with a different unicast destination address. Note that there may be multiple copies traveling between two routers. For example, when a person sends an e-mail message to a group of people, this is multiple unicasting. The e-mail software creates replicas of the message, each with a different destination address, and sends them one by one.

### *Emulation of Multicasting with Unicasting*

You might wonder why we have a separate mechanism for multicasting, when it can be emulated with unicasting. There are several reasons for this; two are obvious:

1. Multicasting is more efficient than multiple unicasting. In Figure 12.3, we can see how multicasting requires less bandwidth than multiple unicasting. In multiple unicasting, some of the links must handle several copies.
2. In multiple unicasting, the packets are created by the source with a relative delay between packets. If there are 1,000 destinations, the delay between the first and the last packet may be unacceptable. In multicasting, there is no delay because only one packet is created by the source.

**Emulation of multicasting through multiple unicasting is not efficient and may create long delays, particularly with a large group.**

### *Multicast Applications*

Multicasting has many applications today such as access to distributed databases, information dissemination, teleconferencing, and distance learning.

**Access to Distributed Databases** Most of the large databases today are distributed. That is, the information is stored in more than one location, usually at the time of production. The user who needs to access the database does not know the location of the information. A user's request is multicast to all the database locations, and the location that has the information responds.

**Information Dissemination** Businesses often need to send information to their customers. If the nature of the information is the same for each customer, it can be multicast. In this way a business can send one message that can reach many customers. For example, a software update can be sent to all purchasers of a particular software package.

**Dissemination of News** In a similar manner news can be easily disseminated through multicasting. One single message can be sent to those interested in a particular topic. For example, the statistics of the championship high school basketball tournament can be sent to the sports editors of many newspapers.

**Teleconferencing** *Teleconferencing* involves multicasting. The individuals attending a teleconference all need to receive the same information at the same time. Temporary or permanent groups can be formed for this purpose. For example, an engineering group that holds meetings every Monday morning could have a permanent group while the group that plans the holiday party could form a temporary group.

**Distance Learning** One growing area in the use of multicasting is *distance learning.* Lessons taught by one single professor can be received by a specific group of students. This is especially convenient for those students who find it difficult to attend classes on campus.

## Broadcasting

In broadcast communication, the relationship between the source and the destination is one to all. There is only one source, but all of the other hosts are the destinations. The Internet does not explicitly support *broadcasting* because of the huge amount of traffic it would create and because of the bandwidth it would need. Imagine the traffic generated in the Internet if one person wanted to send a message to everyone else connected to the Internet.

# 12.2 MULTICAST ADDRESSES

A multicast address is a destination address for a group of hosts that have joined a multicast group. A packet that uses a multicast address as a destination can reach all members of the group unless there are some filtering restriction by the receiver.

## Multicast Addresses in IPv4

Although we have multicast addresses in both data link and network layers, in this section, we discuss the multicast addresses in the network layer, in particular the multicast addresses used in the IPv4 protocol. Multicast addresses for IPv6 are discussed in Chapter 26. In classful addressing, multicast addresses occupied the only single block in class D. In classless addressing the same block has been used for this purpose. In other words, the block assigned for multicasting is 224.0.0.0/4. This means that the block has $2^{28}$ = 268,435,456 addresses (224.0.0.0 to 239,255.255.255). This large block, or the multicast address space as sometimes it is referred to, is divided into some smaller ranges, as shown in Table 12.1, based on RFC 3171. They may change in the future. However, we cannot assign CIDR (slash notation) to every designated range because the division of the main block to small blocks (or subblocks) is not done according to the rules we defined in Chapter 5. These ranges may be subdivided in the future to follow the rules.

**Table 12.1** *Multicast Address Ranges*

| *CIDR* | *Range* | *Assignment* |
|---|---|---|
| 224.0.0.0/24 | 224.0.0.0 → 224.0.0.255 | Local Network Control Block |
| 224.0.1.0/24 | 224.0.1.0 → 224.0.1.255 | Internetwork Control Block |
| | 224.0.2.0 → 224.0.255.255 | AD HOC Block |
| 224.1.0.0/16 | 224.1.0.0 → 224.1.255.255 | ST Multicast Group Block |
| 224.2.0.0/16 | 224.2.0.0 → 224.2.255.255 | SDP/SAP Block |
| | 224.3.0.0 → 231.255.255.255 | Reserved |
| 232.0.0.0/8 | 232.0.0.0 → 224.255.255.255 | Source Specific Multicast (SSM) |
| 233.0.0.0/8 | 233.0.0.0 → 233.255.255.255 | GLOP Block |
| | 234.0.0.0 → 238.255.255.255 | Reserved |
| 239.0.0.0/8 | 239.0.0.0 → 239.255.255.255 | Administratively Scoped Block |

### *Local Network Control Block*

The first block is called local network control block (224.0.0.1/24). The addresses in this block are used for protocol control traffic. In other words, they are not used for general multicast communication. Some multicast or multicast-related protocols use these addresses. The IP packet with the destination address in this range needs to have the value of TTL set to 1, which means that the routers are not allowed to forward these packets. The packet remains in the network, which means two or more networks can use the same address at the same time, a sense of privacy. Table 12.2 shows the assignment of some of these addresses.

### *Internetwork Control Block*

The block 224.0.1/24 is called the Internetwork Control Block. The addresses in this block are also used for protocol control traffic, but the IP packets with one of these addresses as destination can be forwarded by router thought the whole Internet. For example, the address 224.0.1.1 is used by the NTP protocol.

**Table 12.2** *Some addresses in Network Control Block*

| *Address* | *Assignment* |
|---|---|
| 224.0.0.0 | Base address (reserved) |
| 224.0.0.1 | All systems (hosts or routers) on this network |
| 224.0.0.2 | All routers on this network |
| 224.0.0.4 | DMVRP routers |
| 224.0.0.5 | OSPF routers |
| 224.0.0.7 | ST (stream) routers |
| 224.0.0.8 | ST (stream) hosts |
| 224.0.0.9 | RIP2 routers |
| 224.0.0.10 | IGRP routers |
| 224.0.0.11 | Mobile Agents |
| 224.0.0.12 | DHCP servers |
| 224.0.0.13 | PIM routers |
| 224.0.0.14 | RSVP encapsulation |
| 224.0.0.15 | CBT routers |
| 224.0.0.22 | IGMPv3 |

### *AD-HOC Block*

The block 224.0.2.0 to 224.0.255.0 is called AD-HOC Block by IANA. This block was traditionally assigned to some applications that do not fit in the first or second block discussed above. For example, the block 224.0.18/24 is assigned to Dow Jones. Note that the range of this block has two unusual features. First, the last address should be 224.2.255.255. Second, the whole block cannot be represented in CIDR notation (the block allocation does not follow the rules of classless addressing we discussed in Chapter 5).

### *Stream Multicast Group Block*

The block 224.1.0.0/16 is called Stream Multicast Group Block and is allocated for stream multimedia.

### *SAP/SDP Block*

The block 224.2.0.0/16 is used for Session Announcement Protocol and Session Directory Protocol (RFC 2974).

### *SSM Block*

The block 232.0.0.0/8 is used for Source Specific Multicasting. We discuss SSS later in the chapter, when we introduce IGMPv3.

### *GLOP Block*

The block 233.0.0.0/8 is called the GLOP block (not an acronym nor an abbreviation). This block defines a range of globally assigned addresses that can be use inside an

autonomous system (AS). As we learned in Chapter 11, each autonomous system is assigned a 16-bit number. One can insert the AS number as the two middle octet in the block to create a range of 256 multicast addresses (233.x.y.0 to 233.x.y.255), in which x.y is the AS number.

### *Administratively Scoped Block*

The block 239.0.0.0/8 is called the Administratively Scoped Block. The addresses in this block are used in a particular area of the Internet. The packet whose destination address belongs to this range is not supposed to leave the area. In other words, an address in this block is restricted to an organization.

### *Netstat Utility*

The *netstat utility* can be used to find the multicast addresses supported by an interface.

### Example 12.1

We use *netstat* with three options, -n, -r, and -a. The -n option gives the numeric versions of IP addresses, the -r option gives the routing table, and the -a option gives all addresses (unicast and multicast). Note that we show only the fields relative to our discussion.

```
$netstat -nra
Kernel IP routing table
```

| Destination | Gateway | Mask | Flags | Iface |
|---|---|---|---|---|
| 153.18.16.0 | 0.0.0.0 | 255.255.240.0 | U | eth0 |
| 169.254.0.0 | 0.0.0.0 | 255.255.0.0 | U | eth0 |
| 127.0.0.0 | 0.0.0.0 | 255.0.0.0 | U | l0 |
| **224.0.0.0** | **0.0.0.0** | **224.0.0.0** | **U** | **eth0** |
| 0.0.0.0 | 153.18.31 | 0.0.0.0 | UG | eth0 |

Note that the multicast address is shown in color. Any packet with a multicast address from 224.0.0.0 to 239.255.255.255 is masked and delivered to the Ethernet interface.

## Selecting Multicast Address

To select a multicast address to be assigned to a group is not an easy task. The selection of address depends on the type of application. Let us discuss some cases.

### *Limited Group*

The administrator can use the AS number (x.y) and choose an address between 239.x.y.0 and 239.x.y.255 (Administratively Scoped Block) that is not used by any other group as the multicast address for that particular group. For example, assume a college professor needs to create a group address to communicate with her students. If the AS number that the college belongs to is 23452, which can be written as $(91.156)_{256}$, this gives the college a range of 256 addresses: 233.91.156.0 to 233.91.156.255. The college administration can grant the professor one of the addresses that is not used in this range, for example, 233.91.156.47. This can become the group address for the professor to use to send multicast addresses to the students. However, the packets cannot go beyond the college AS territory.

### *Larger Group*

If the group is spread beyond an AS territory, the previous solution does not work. The group needs to choose an address from the SSM block (232.0.0.0/8). There is no need to get permission to use an address in this block, because the packets in source-specific multicasting are routed based on the group and the source address; they are unique.

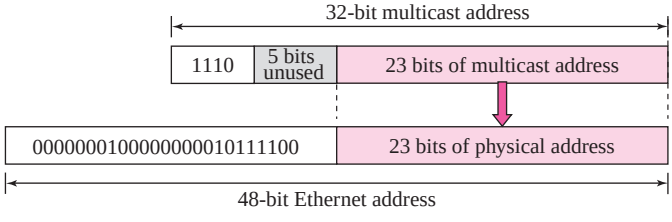
## Delivery of Multicast Packets at Data Link Layer

Because the IP packet has a multicast IP address, the ARP protocol cannot find the corresponding MAC (physical) address to forward the packet at the data link layer. What happens next depends on whether or not the underlying data link layer supports physical multicast addresses.

### *Network with Multicast Support*

Most LANs support physical multicast addressing. Ethernet is one of them. An Ethernet physical address (MAC address) is six octets (48 bits) long. If the first 25 bits in an Ethernet address are 00000001 00000000 01011110 0, this identifies a physical multicast address for the TCP/IP protocol. The remaining 23 bits can be used to define a group. To convert an IP multicast address into an Ethernet address, the multicast router extracts the least significant 23 bits of a multicast IP address and inserts them into a multicast Ethernet physical address (see Figure 12.4).

**Figure 12.4** *Mapping class D to Ethernet physical address*




However, the group identifier of a multicast address block in $IPv_4$ address is 28 bits long, which implies that 5 bits are not used. This means that 32 ($2^5$) multicast addresses at the IP level are mapped to a single multicast address. In other words, the mapping is many to one instead of one to one. If the 5 leftmost bits of the group identifier of a multicast address are not all zeros, a host may receive packets that do not really belong to the group in which it is involved. For this reason, the host must check the IP address and discard any packets that do not belong to it.

**An Ethernet multicast physical address is in the range**
**01:00:5E:00:00:00 to 01:00:5E:7F:FF:FF.**

### Example 12.2

Change the multicast IP address 232.43.14.7 to an Ethernet multicast physical address.

**Solution**

We can do this in two steps:

**a.** We write the rightmost 23 bits of the IP address in hexadecimal. This can be done by changing the rightmost 3 bytes to hexadecimal and then subtracting 8 from the leftmost digit if it is greater than or equal to 8. In our example, the result is 2B:0E:07.

**b.** We add the result of part a to the starting Ethernet multicast address, which is 01:00:5E:00:00:00. The result is

**01:00:5E:2B:0E:07**

### Example 12.3

Change the multicast IP address 238.212.24.9 to an Ethernet multicast address.

**Solution**

**a.** The rightmost 3 bytes in hexadecimal are D4:18:09. We need to subtract 8 from the leftmost digit, resulting in 54:18:09.

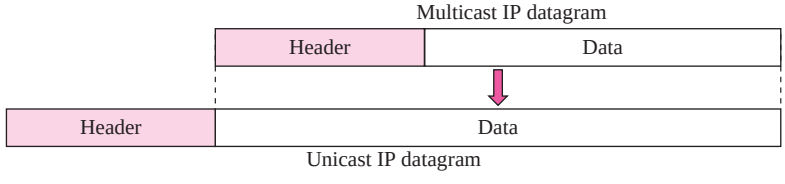
**b.** We add the result of part a to the Ethernet multicast starting address. The result is

**01:00:5E:54:18:09**

### *Network with No Multicast Support*

Most WANs do not support physical multicast addressing. To send a multicast packet through these networks, a process called *tunneling* is used. In **tunneling,** the multicast packet is encapsulated in a unicast packet and sent through the network, where it emerges from the other side as a multicast packet (see Figure 12.5).

**Figure 12.5** *Tunneling*



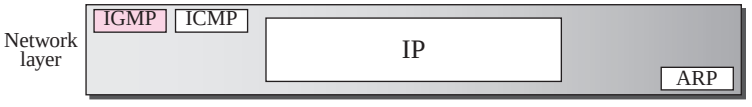

## 12.3 IGMP

Multicast communication means that a sender sends a message to a group of recipients that are members of the same group. Since one copy of the message is sent by the sender, but copied and forwarded by routers, each multicast router needs to know the list of groups that have at least one loyal member related to each interface. This means that the multicast routers need to collect information about members and share it with other multicast routers. Collection of this type of information is done at two levels:

locally and globally. A multicast router connected to a network is responsible to collect this type of information locally; the information collected can be globally propagated to other routers. The first task is done by the IGMP protocol; the second task is done by the multicast routing protocols. We first discuss IGMP in this section.

The **Internet Group Management Protocol (IGMP)** is responsible for correcting and interpreting information about group members in a network. It is one of the protocols designed at the IP layer for this purpose. Figure 12.6 shows the position of the IGMP protocol in relation to other protocols in the network layer.

**Figure 12.6** *Position of IGMP in the network layer*




## Group Management

IGMP is not a multicasting routing protocol; it is a protocol that manages *group membership*. In any network, there are one or more multicast routers that distribute multicast packets to hosts or other routers. The IGMP protocol gives the *multicast routers* information about the membership status of hosts (routers) connected to the network.

A multicast router may receive thousands of multicast packets every day for different groups. If a router has no knowledge about the membership status of the hosts, it must forward all of these packets. This creates a lot of traffic and consumes bandwidth. A better solution is to keep a list of groups in the network for which there is at least one loyal member. IGMP helps the multicast router create and update this list.

**IGMP is a group management protocol. It helps a multicast router create and update a list of loyal members related to each router interface.**

IGMP has gone through three versions. Versions 1 and 2 provide what is called any-source multicast (ASM), which means that the group members receive a multicast message no matter where it comes from. The IGMP version 3 provides what is called source-specific multicast (SSM), which means that the recipient can choose to receive multicast messages coming from a list of predefined sources. In this section we discuss only IGMPv3.

## IGMP Messages

IGMPv3 has two types of messages: *membership query message* and *membership report message*. The first type can be used in three different formats: *general*, *group-specific*, and *group-and-source-specific*, as shown in Figure 12.7.

### *Membership Query Message Format*

A membership query message is sent by a router to find active group members in the network. Figure 12.8 shows the format of this message.

**Figure 12.7** *IGMP messages*

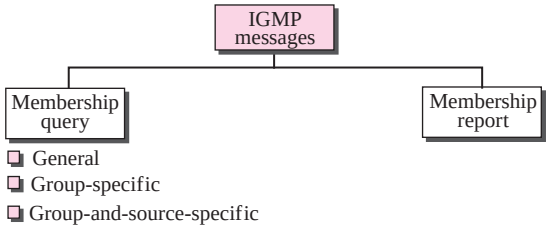
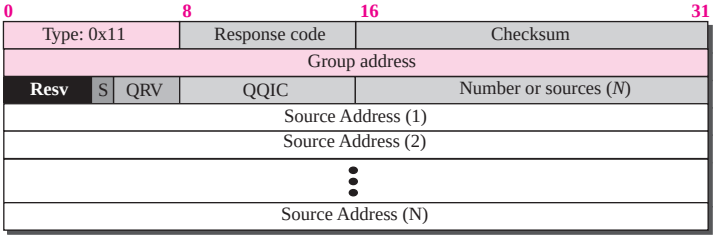



**Figure 12.8** *Membership query message format*



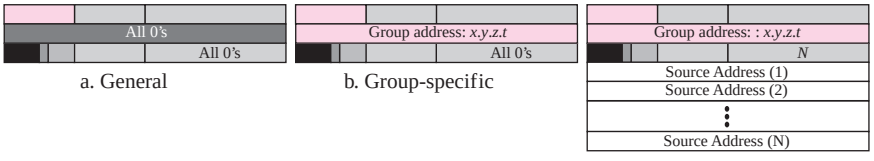

A brief description of each field follows:

- **Type.** This 8-bit field defines the type of the message. The value is 0X11 for a membership query message.
- **Maximum Response Code.** This 8-bit field is used to define the *response time* of a recipient of the query as we will show shortly.
- **Checksum.** This is a 16-bit field holding the checksum. The checksum is calculated over the whole IGMP message.
- **Group Address.** This 32-bit field is set to 0 in a general query message; it is set to IP multicast being queried when sending a *group-specific* or *group-and-source-specific* query message.
- **Resv.** This 4-bit field is reserved for the future and it is not used.
- **S.** This is a 1-bit *suppress flag*. When this field is set to 1, it means that the receivers of the query message should suppress the normal timer updates.
- **QRV.** This 3-bit field is called *querier's robustness variable*. It is used to monitor the robustness in the network.
- **QQIC.** This 8-bit field is called *querier's query interval code*. This is used to calculate the querier's query interval (QQI), as we will show shortly.
- **Number of sources (N).** This 16-bit field defines the number of 32-bit unicast source addresses attached to the query. The value of this field is zero for the *general query* and the *group-specific query,* and nonzero in the *group-and-source-specific query.*
- **Source Addresses.** These multiple 32-bit fields list the *N* source addresses, the origin of multicast messages. The value of N is defined in the previous field.

### Three Formats of Query Messages

As mentioned in the previous sections, there are three formats for query messages: *general* query, *group-specific* query, and *group-and-source-specific* query. Each format is used for a different purpose. Figure 12.9 shows one example of these three types of messages to be compared.

**Figure 12.9** *Three formats of query messages*




a. General b. Group-specific c. Group-and-source-specific

**a.** In a *general query message*, the querier router probes each neighbor to report the whole list of its group membership (interest in any multicast group).

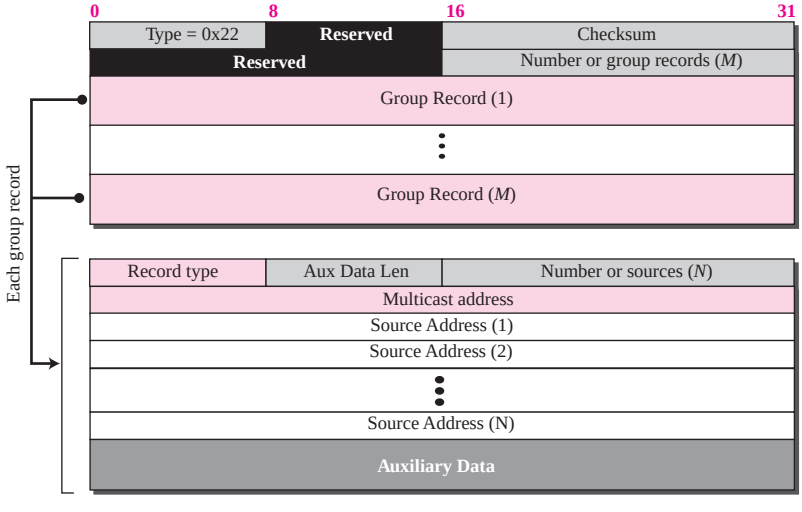
**b.** In a *group-specific query message*, the querier router probes each neighbor to report if it is still interested in a specific multicast group. The multicast group address is defined as *x.y.z.t* in the group address field of the query.

**c.** In a *group-and-source-specific query message*, the querier router probes each neighbor to report if it is still in a specific multicast group, *x.y.z.t*, coming from any of the *N* sources whose unicast addresses are defined in this packet.

### Membership Report Message Format

Figure 12.10 shows the format of an IGMP membership report message format.

**Figure 12.10** *Membership report message format*



- **Type.** This 8-bit field with the value 0x22 defines the type of the message.
- **Checksum.** This is a 16-bit field carrying the checksum. The checksum is calculated over the entire IGMP message.
- **Number of Group Records (M).** This 16-bit field defines the number of group records carried by the packet.
- **Number of Group Records.** There can be zero or more group records of variable length. The fields of each group record will be explained below.

Each group record includes the information related to the responder's membership in a single multicast group. The following briefly describes each field:

- **Record Type.** Currently there are six record types as shown in Table 12.3. We learn about their applications shortly.

**Table 12.3** *Record Type*

| *Category* | *Type* | *Type Value* |
|---|---|---|
| Current-State-Record | Mode_Is_Include | 1 |
| | Mode_Is_Exclude | 2 |
| Filter-Mode-Change-Record | Change_To_Include_Mode | 3 |
| | Change_To_Exclude_Mode | 4 |
| Source-List-Change-Record | Allow_New_Sources | 5 |
| | Block_Old_Sources | 6 |

- **Aux Data Len.** This 8-bit field defines the length of the auxiliary data included in each group record. If the value of this field is zero, it means there is no auxiliary data included in the packet. If the value is nonzero, it specifies the length of auxiliary data in words of 32 bits.
- **Number of Sources (N).** This 16-bit field defines the number of 32-bit multicast source addresses attached to the report.
- **Source Addresses.** These multiple 32-bit fields list the *M* source addresses. The value of *M* is defined in the previous field.
- **Aux Data.** This field contains any auxiliary data that may be included in the report message. The IGMP has not yet defined any auxiliary data, but it may be added to the protocol in the future.

## IGMP Protocol Applied to Host

In this section, we discuss how a host implements the protocol.

### *Socket State*

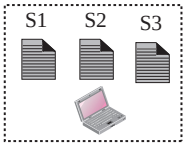
The management of groups starts with the processes (running application programs) on a host connected to an interface. Each process, which is associated with a *socket* as we will see in Chapter 17, has a record for each multicast group from which the socket wishes to receive a multicast message. The record also shows one of the two modes: *include* mode or *exclude* mode. If the record is in *include* mode, it lists the unicast source addresses from which the socket accepts the group messages. If the record is in

*exclude* mode, it lists the unicast source addresses that the socket will not accept the group messages. In other words, the include mode means "only … ," the exclude mode means "all, but … ." The state can be organized in a table, in which a row defines one single record. A socket may have more than one record if it expects to receive multicast messages destined for more than one group.

### Example 12.4

Figure 12.11 shows a host with three processes: S1, S2, and S3. The first process has only one record; the second and the third processes each have two records. We have used lowercase alphabet to show the source address.

**Figure 12.11** *Socket states*




**Legend** S: Socket, a, b, ...: Source addresses

States Table

| Socket | Multicast group | Filter | Source addresses |
|---|---|---|---|
| S1 | 226.14.5.2 | Include | a, b, d, e |
| S2 | 226.14.5.2 | Exclude | a, b, c |
| S2 | 228.24.21.4 | Include | b, c, f |
| S3 | 226.14.5.2 | Exclude | b, c, g |
| S3 | 228.24.21.4 | Include | d, e, f |

### *Interface State*

As Figure 12.11, shows there can be overlap information in the socket records. Two or more sockets may have a record with the same multicast groups. To be efficient, group management requires that the interface connecting the host to the network also keep an interface state. The interface state is originally empty, but it will build up when socket records are changed (creation in this sense also means change). However, the interface state keeps only one record for each multicast group. If new socket records are created for the same multicast group, the corresponding interface will changed to reflect the new change. For example, the interface state corresponding to socket state in Figure 12.11 needs to have only two records, instead of five, because only two multicast groups are involved. The interface state, however, needs to keep an interface timer for the whole state and one timer for each record. We discuss the application of these timers shortly. The only problem in combining records is the list of resources. If a record with the same multicast group has two or more different lists of resources, the following two rules need to be followed to combine the list of resources.

1. If any of the records to be combined has the *exclusive* filter mode, then the resulting interface record will have the *exclusive* filter mode and the list of the source addresses is made as shown below:
   - **a.** Apply the set intersection operation on all the address lists with *exclusive* filters.
   - **b.** Apply the set difference operation on the result of part *a* and all the address lists with *inclusive* filters.

2. If all the records to be combined have the *inclusive* filter mode, then the resulting interface record will have the *inclusive* filter mode and the list of the source addresses is found by applying the set union operations on all the address lists.

**Each time there is a change in any socket record, the interface state will change using the above-mentioned rules.**

### Example 12.5

We use the two rules described above to create the interface state for the host in Example 12.4. First we found the list of source address for each multicast group.

a. Multicast group 226.14.5.2 has two *exclude* lists and one *include* list. The result is an *exclude* list as calculated below. We use the dot sign for intersection operation and minus sign for the difference operation.

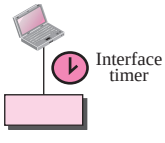
$$\text{exclude source list} = \{a, b, c\} \cdot \{b, c, g\} - \{a, b, d, e\} = \{c\}$$

b. Multicast group: 228.24.21.4 has two *include* lists. The result is an include list as calculated below. We use the plus sign for the union operation.

$$\text{include source list} = \{b, c, f\} + \{d, e, f\} = \{b, c, d, e, f\}$$

Figure 12.12 shows the interface state. The figure shows that there is one timer for the interface, but each state related to each multicast group has its own timer.

**Figure 12.12** *Interface state*




Interface state

| Multicast group | Group timer | Filter | Source addresses |
|---|---|---|---|
| 226.14.5.2 | | Exclude | c |
| 228.24.21.4 | | Include | b, c, d, e, f |

### *Sending Change-State Reports*

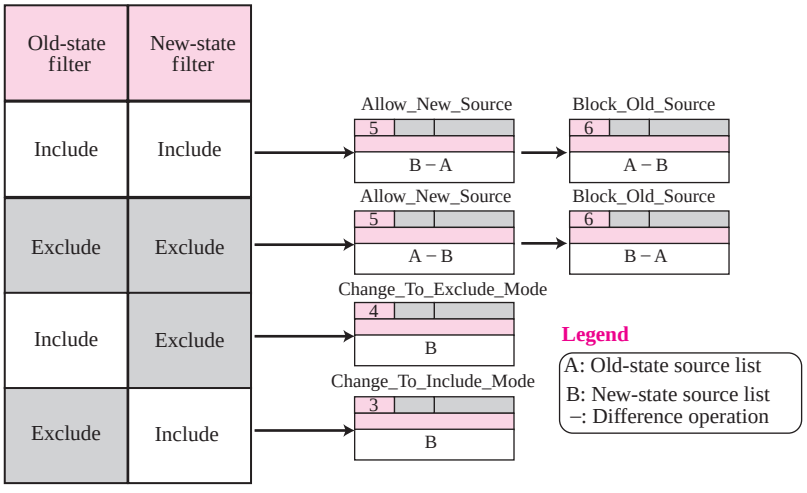
If there is any change in the interface state, the host needs to immediately send a membership report message for that group, using the appropriate group record(s). As Figure 12.13 shows, four different cases may occur in the change, based on the old-state filter and the new state filter. We have shown only the group records, not the whole report.

As the figure shows, in the first two cases, the report contains two group records; in the last two cases, the report contains only one group record.

### *Receiving Query Reports*

When a host receives a query, it does not respond immediately; it delays the response by a random amount of time calculated from the value of the Max Resp Code field as

**Figure 12.13** *Sending change state reports*




described later. The action of the host depends on the type of the query received as shown below:

1. If the received query is a general query, the host reset the interface timer (see Figure 12.12) to the calculated delay value. This means if there is any previous delayed response, it is cancelled.
2. If the received query is a group-specific query, then the corresponding group time (see Figure 12.12) is reset to the shorter value of the remaining time for the timer or the calculated delay. If a timer is not running, its remaining time is considered to be infinity.
3. If the received query is a group-and-source-specific query, then the action is the same as the previous case. In addition, the list of sources is recorded for the delayed response.

### *Timer Expiration*

Membership report messages are sent by a host when a timer expires. However, the types and number of group records contained in the message depends on the timer.

1. If the expired timer is the interface timer set after a general query received, then the host sends one membership report that contains one Current-State-Record for each group in the interface state. The type of each record is either Mode-Is-Include (type 1) or Mode-Is-Exclude (type 2) depending on the filtering mode of the group. However, if all records to be sent do not fit in one report, they can be split into several reports.
2. If the expired timer is the group timer set after a group-specific query received (which means that the interface has recorded no source list for this group), then the host sends one membership report that contains only one Current-State-Record for that particular group if, and only if, the group is still active. The single record contained in the report is of type Mode-Is-Include (type 1) or Mode-Is-Exclude (type 2).

3. If the expired timer is the group timer set after a group-and-source-specific query received (which means that the interface has a recorded source list for this group), then the host sends one membership report that contains only one Current-State-Record for that particular group if, and only if, the group is still active. The type of the single record contained in the report and source list depends on the filter mode of the group:
   a. If the group filter is include, the record type is Mode_Is_Include (type 1) and the source list is (A . B) in which A is the group source list and B is the received source list. The dot sign means the intersection operation.
   b. If the group filter is exclude, the record type is Mode_Is_Exclude (type 2) and the source list is (B – A) in which A is the group source list and B is the received source list. The minus sign means the difference operation.

### *Report Suppression*

In previous versions of IGMP, if a host receives a report sent by another host, it cancelled the corresponding timer in its interface state, which means suppressing the pending report. In IGMPv3, this mechanism has been removed from the protocol for some practical reasons described in RFC 3376.

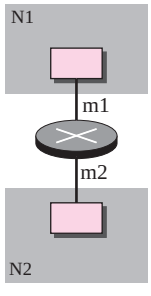
## IGMP Protocol Applied to Router

In this section, we try to discuss how a router implements the protocol. The duty of a multicast router, which is often called a *querier* in IGMPv3, is much more complex than the one for previous versions. A querier needs to handle six types of group records contained in a membership report.

### *Querier's State*

The multicast router needs to maintain the state information for each multicast group associated with each network interface. The state information for each network interface is a table with each row related to a multicast group. The information for each multicast group consists of the multicast address, group timer, filter mode, and source records. However, each source code includes the address of the source and a corresponding timer. Figure 12.14 shows an example of a multicast router and its two state tables, one related to interface m1 and the other to interface m2.

**Figure 12.14** *Router states*




State for interface m1

| Multicast group | Timer | Filter | Source addresses |
|---|---|---|---|
| 227.12.15.21 | | Exclude | (a , ) (c , ) |
| 228.21.25.41 | | Include | (b , ) (d , ) (e , ) |

State for interface m2

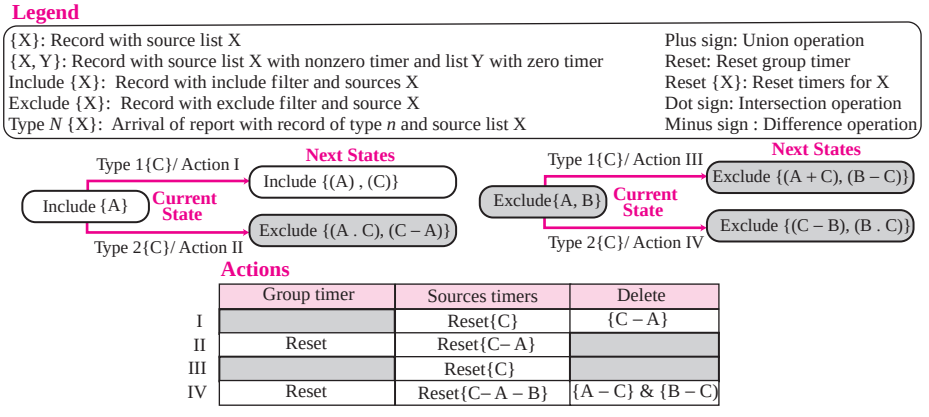
| Multicast group | Timer | Filter | Source addresses |
|---|---|---|---|
| 226.10.11.8 | | Exclude | (b , ) |
| 227.21.25.41 | | Include | (a , ) (b , ) (c , ) |
| 228.32.12.40 | | Include | (d , ) (e , ) (f , ) |

### *Action Taken on Membership Report Reception*

A multicast router sends out queries and receives reports. In this section we show the changes in the state of a router when it receives a report.

**Reception of Report in Response to General Query** When a router sends a general query, it expects to receive a report or reports. This type of a report normally contains Current-State-Record (types 1 and 2). When such a report arrives, the router changes its state, as shown in Figure 12.15. The figure shows the state of each record in the router based on the current state and the record arrived in the report. In other words, the figure is a state transition diagram. When a record is extracted from an arrived report, it changes the state of the corresponding record (with the same group address) and invokes some actions. We have shown the four different sets of actions separately, one set for each change of state.

**Figure 12.15** *Change of state related to general query report*

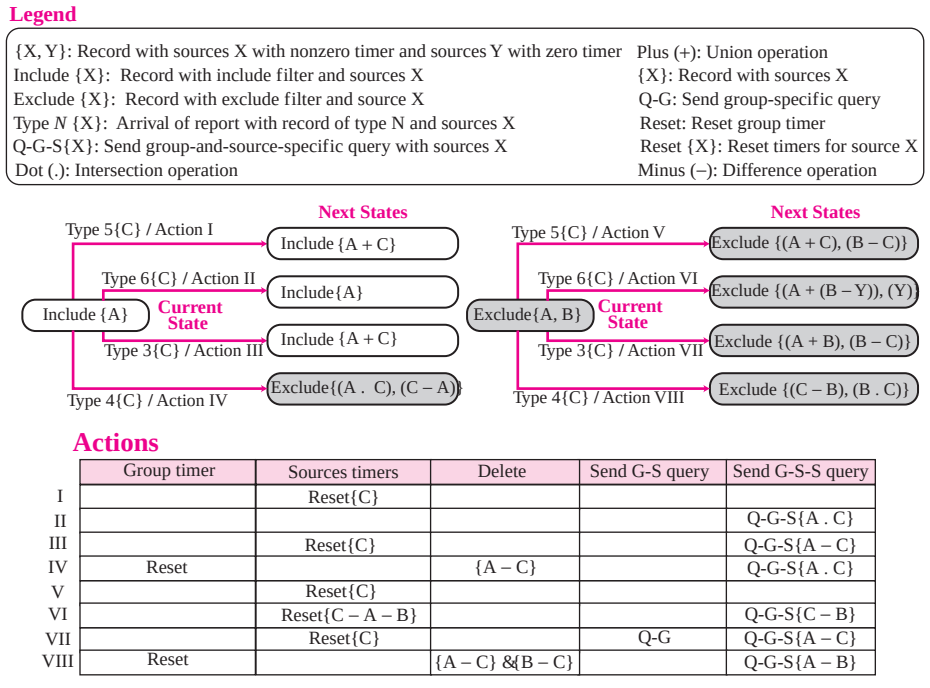



**Reception of Reports in Response to Other Queries** When a router sends a group-specific or group-and-source-specific query, it expects to receive a report or reports. This type of report normally contains Filter_Mode_Change_Record (types 3 and 4) or Source_List_Change Record (types 5 and 6). When such a report arrives, the router changes its state as shown in Figure 12.16. The figure shows the next state of each record in the router based on the current state and the type of the record contained in the arrived report. In other words, the figure is a state transition diagram. We can have eight different cases.

## Role of IGMP in Forwarding

As we discussed before, IGMP is a management protocol. It only collects information to help the router attached to the network to make a decision about forwarding or not forwarding a received packet coming from a specific source and destined for a multicast group. In previous versions of IGMP, the forwarding recommendation was based only on the destination multicast address of a packet; in IGMPv3, the recommendation

**Figure 12.16** *Change of state related to group-specific and group-and-source specific report*




**Actions**

| | Group timer | Sources timers | Delete | Send G-S query | Send G-S-S query |
|---|---|---|---|---|---|
| I | | Reset{C} | | | |
| II | | | | | Q-G-S{A . C} |
| III | | Reset{C} | | | Q-G-S{A – C} |
| IV | Reset | | {A – C} | | Q-G-S{A . C} |
| V | | Reset{C} | | | |
| VI | | Reset{C – A – B} | | | Q-G-S{C – B} |
| VII | | Reset{C} | | Q-G | Q-G-S{A – C} |
| VIII | Reset | | {A – C} &{B – C} | | Q-G-S{A – B} |

is based on both the destination address and the source address. RFC 3376 mentions six different recommendations that IGMPv3 software can give to the IP multicast router to forward or not to forward a packet with respect to an interface. The recommendation is based on the interface state of the router as shown in Figure 12.14. If the multicast destination address does not exist in the router state, the recommendation is definitely not to forward the packet. When the destination address exists in the router state, then the recommendation is based on the filter mode and source address, as shown in Table 12.4.

**Table 12.4** *Forwarding Recommendation of IGMPv3*

| *Filter Mode* | *Source Address* | *Source Timer Value* | *Recommendation* |
|---|---|---|---|
| Include | In the list | greater than zero | **Forward** |
| Include | In the list | zero | Do not forward |
| Include | Not in the list | | Do not forward |
| Exclude | In the list | greater than zero | **Forward** |
| Exclude | In the list | zero | Do not forward |
| Exclude | Not in the list | | **Forward** |

The table shows that it is recommended to forward the multicast IP datagram in three cases. The first case (first row) is obvious; the source address is in the list, which means at least one host in the network desires to receive this type of packet. The third case (row 6) is also clear; the mode is excluded and the source address is not excluded,

which means that at least one host likes to receive group messages from this source. What that may not be clear is the second case (row 4). The filter mode is excluded and source address is in the list, which means that this source should be excluded and the packet should not be normally forwarded. The rationale for forwarding the packet in this case is that the timer for this source has not expired yet. The source is to be excluded when the timer is expired.

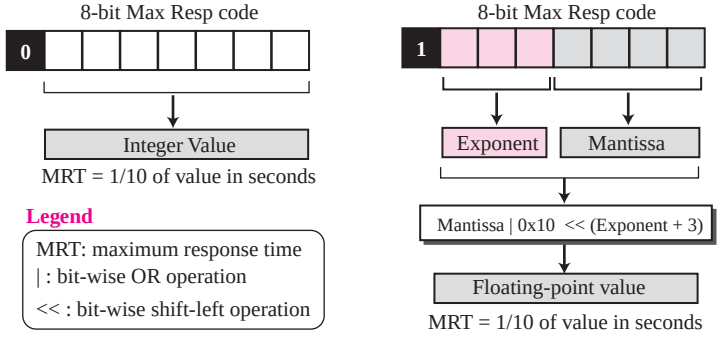
## Variables and Timers

In the previous sections, we have used several variables and timers whose meanings we need to clarify here.

### *Maximum Response Time*

The maximum response time is the maximum time allowed before sending a report in response to a query. It is calculated from the value of the 8-bit Max Resp Code field in the query. In other words, the query defines the maximum response time. The calculation of the maximum response time is shown in Figure 12.17. If the value of Max Resp Code is less than 128 (leftmost bit is zero), the maximum response time is an integer value; if the value of Max Resp Code is greater than or equal 128 (leftmost bit is 1), the maximum response time is a floating-point value.

**Figure 12.17** *Calculation of maximum response time*




### *Querier's Robustness Variable (QRV)*

IGMP monitors the packet loss in the network (by measuring the delays in receiving the reports) and adjusts the value of the QRV. The value of this variable dictates how many times a response message to a query should be sent. If the network loses more packets, the router sets a higher value for QRV. The number of times a host should send a response to a query is QRV −1. The default value is 2, which means that the host needs to send the response at least once. If the router has specified a value of 5, it means that the router needs to send the response four times. Note that the value of QRV should never be 0 or 1; otherwise the default value 2 is assumed.

#### *Querier's Query Interval*

This is the interval between the general queries. The default value is 125. This default value can be changed by an administrator to control the traffic on the network.

#### *Other Variables and Timers*

There are several other variables and timers defined in RFC 3376 which is more interesting to the administrator. We recommend this RFC for more details.

### Encapsulation

The IGMP message is encapsulated in an IP datagram with the value of protocol field set to 2 and the TTL field set to 1. The destination IP address of datagram, however, depends on the type of the message, as shown in Table 12.5.

**Table 12.5** *Destination IP Addresses*

| *Message Type* | *IP Address* |
|---|---|
| General Query | 224.0.0.1 |
| Other Queries | Group address |
| Report | 224.0.0.22 |

### Compatibility with Older Versions

To be compatible with version 1 and 2, IGMPv3 software is designed to accept messages defined in version 1 and 2. Table 12.6 shows the value of type fields and the type of messages in versions 1 and 2.

**Table 12.6** *Messages in Versions 1 and 2*

| *Version* | *Type Value* | *Message Type* |
|---|---|---|
| 1 | 0x11 | Query |
| | 0x12 | Membership Report |
| 2 | 0x11 | Query |
| | 0x16 | Membership Report |
| | 0x17 | Leave Group |

## 12.4 MULTICAST ROUTING

Now we show how information collected by IGMP is disseminated to other routers using multicast routing protocols. However, we first discuss the idea of optimal routing, common in all multicast protocols. We then give an overview of multicast routing protocols.

### Optimal Routing: Shortest Path Trees

The process of optimal interdomain routing eventually results in the finding of the **shortest path tree.** The root of the tree is the source and the leaves are the potential destinations. The path from the root to each destination is the shortest path. However,

the number of trees and the formation of the trees in unicast and multicast routing are different. Let us discuss each separately.

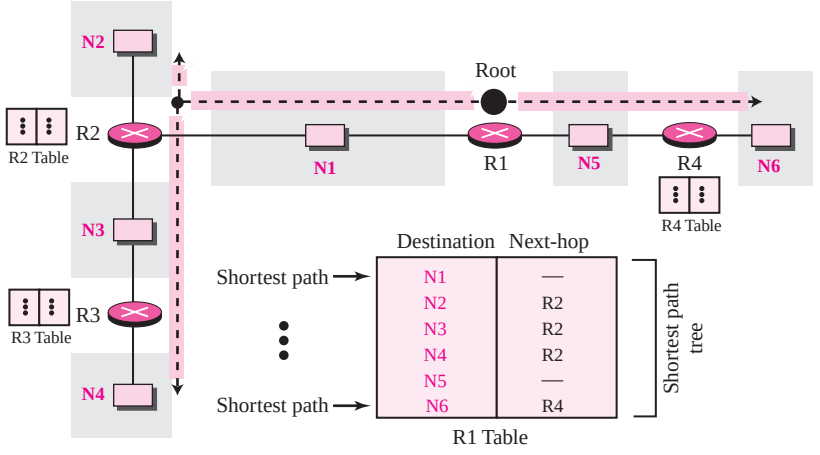
### Unicast Routing

In unicast routing, when a router receives a packet to forward, it needs to find the shortest path to the destination of the packet. The router consults its routing table for that particular destination. The next-hop entry corresponding to the destination is the start of the shortest path. The router knows the shortest path for each destination, which means that the router has a shortest path tree to optimally reach all destinations. In other words, each line of the routing table is a shortest path; the whole routing table is a shortest path tree. In unicast routing, each router needs only one shortest path tree to forward a packet; however, each router has its own shortest path tree. Figure 12.18 shows the situation.

The figure shows the details of the routing table and the shortest path tree for router R1. Each line in the routing table corresponds to one path from the root to the corresponding network. The whole table represents the shortest path tree.

**In unicast routing, each router in the domain has a table that defines a shortest path tree to possible destinations.**

**Figure 12.18** *Shortest path tree in unicast routing*




### Multicast Routing

When a router receives a multicast packet, the situation is different. A multicast packet may have destinations in more than one network. Forwarding of a single packet to members of a group requires a shortest path tree. If we have *n* groups, we may need *n* shortest path trees. We can imagine the complexity of multicast routing. Two approaches have been used to solve the problem: source-based trees and group-shared trees.

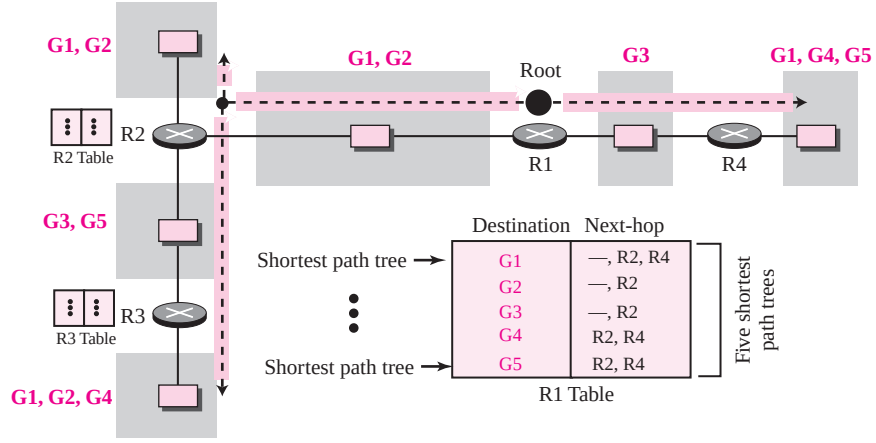
**In multicast routing, each involved router needs to construct a shortest path tree for each group.**

**Source-Based Tree** In the **source-based tree** approach, each router needs to have one shortest path tree for each group. The shortest path tree for a group defines the next hop for each network that has loyal member(s) for that group. In Figure 12.19, we assume that we have only five groups in the domain: G1, G2, G3, G4, and G5. At the moment G1 has loyal members in four networks, G2 in three, G3 in two, G4 in two, and G5 in two. We have shown the names of the groups with loyal members on each network. The figure also shows the multicast routing table for router R1. There is one shortest path tree for each group; therefore there are five shortest path trees for five groups. If router R1 receives a packet with destination address G1, it needs to send a copy of the packet to the attached network, a copy to router R2, and a copy to router R4 so that all members of G1 can receive a copy.

In this approach, if the number of groups is *m*, each router needs to have *m* shortest path trees, one for each group. We can imagine the complexity of the routing table if we have hundreds or thousands of groups. However, we will show how different protocols manage to alleviate the situation.

**In the source-based tree approach, each router needs to have one shortest path tree for each group.**

**Figure 12.19** *Source-based tree approach*

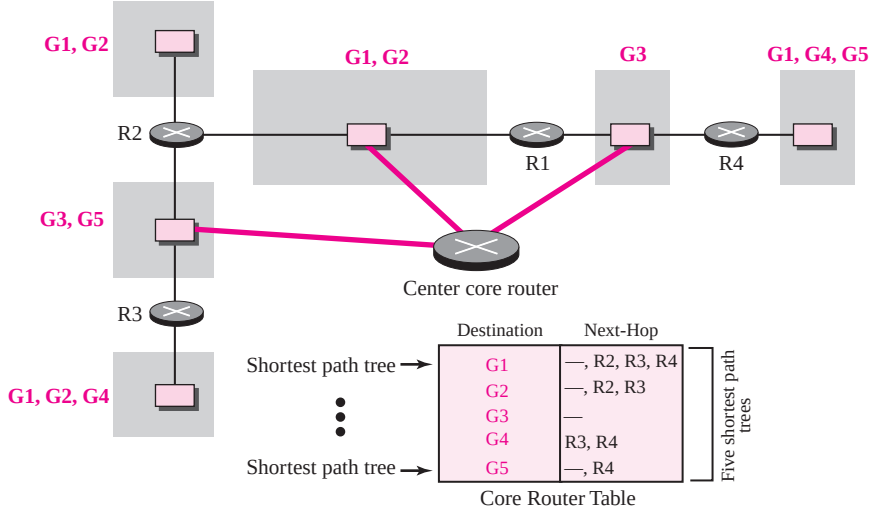



**Group-Shared Tree** In the **group-shared tree** approach, instead of each router having *m* shortest path trees, only one designated router, called the center core, or rendezvous router, takes the responsibility of distributing multicast traffic. The core has *m* shortest path trees in its routing table. The rest of the routers in the domain have none.

If a router receives a multicast packet, it encapsulates the packet in a unicast packet and sends it to the core router. The core router removes the multicast packet from its capsule, and consults its routing table to route the packet. Figure 12.20 shows the idea.

**In the group-shared tree approach, only the core router, which has a shortest path tree for each group, is involved in multicasting.**

**Figure 12.20** *Group-shared tree approach*




## 12.5 ROUTING PROTOCOLS

During the last few decades, several multicast routing protocols have emerged. Some of these protocols are extensions of unicast routing protocols; some are totally new. We discuss these protocols in the remainder of this chapter. Figure 12.21 shows the taxonomy of these protocols.

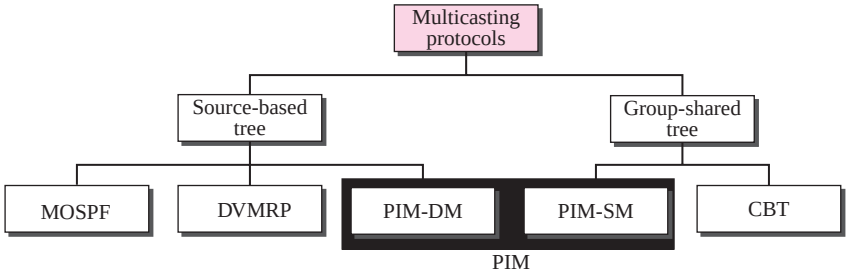
### Multicast Link State Routing: MOSPF

In this section, we briefly discuss multicast link state routing and its implementation in the Internet, MOSPF.

#### *Multicast Link State Routing*

We discussed unicast link state routing in Chapter 11. We said that each router creates a shortest path tree using Dijkstra's algorithm. The routing table is a translation of the shortest path tree. Multicast link state routing is a direct extension of unicast routing and uses a source-based tree approach. Although unicast routing is quite involved, the extension to multicast routing is very simple and straightforward.

**Figure 12.21** *Taxonomy of common multicast protocols*




**Multicast link state routing uses the source-based tree approach.**

Recall that in unicast routing, each node needs to advertise the state of its links. For multicast routing, a node needs to revise the interpretation of *state*. A node advertises every group that has any loyal member on the link. Here the meaning of state is "what groups are active on this link." The information about the group comes from IGMP (discussed earlier in the chapter). Each router running IGMP solicits the hosts on the link to find out the membership status.

When a router receives all these LSPs, it creates *n* (*n* is the number of groups) topologies, from which *n* shortest path trees are made using Dijkstra's algorithm. So each router has a routing table that represents as many shortest path trees as there are groups.

The only problem with this protocol is the time and space needed to create and save the many shortest path trees. The solution is to create the trees only when needed. When a router receives a packet with a multicast destination address, it runs the Dijkstra algorithm to calculate the shortest path tree for that group. The result can be cached in case there are additional packets for that destination.

### *MOSPF*

**Multicast Open Shortest Path First (MOSPF)** protocol is an extension of the OSPF protocol that uses multicast link state routing to create source-based trees. The protocol requires a new link state update packet to associate the unicast address of a host with the group address or addresses the host is sponsoring. This packet is called the group-membership LSA. In this way, we can include in the tree only the hosts (using their unicast addresses) that belong to a particular group. In other words, we make a tree that contains all the hosts belonging to a group, but we use the unicast address of the host in the calculation. For efficiency, the router calculates the shortest path trees on demand (when it receives the first multicast packet). In addition, the tree can be saved in cache memory for future use by the same source/group pair. MOSPF is a **data-driven** protocol; the first time an MOSPF router sees a datagram with a given source and group address, the router constructs the Dijkstra shortest path tree.

## Multicast Distance Vector

In this section, we briefly discuss multicast distance vector routing and its implementation in the Internet, DVMRP.

### *Multicast Distance Vector Routing*

Unicast distance vector routing is very simple; extending it to support multicast routing is complicated. Multicast routing does not allow a router to send its routing table to its neighbors. The idea is to create a table from scratch using the information from the unicast distance vector tables.

Multicast distance vector routing uses source-based trees, but the router never actually makes a routing table. When a router receives a multicast packet, it forwards the packet as though it is consulting a routing table. We can say that the shortest path tree is evanescent. After its use (after a packet is forwarded) the table is destroyed.

To accomplish this, the multicast distance vector algorithm uses a process based on four decision-making strategies. Each strategy is built on its predecessor. We explain them one by one and see how each strategy can improve the shortcomings of the previous one.

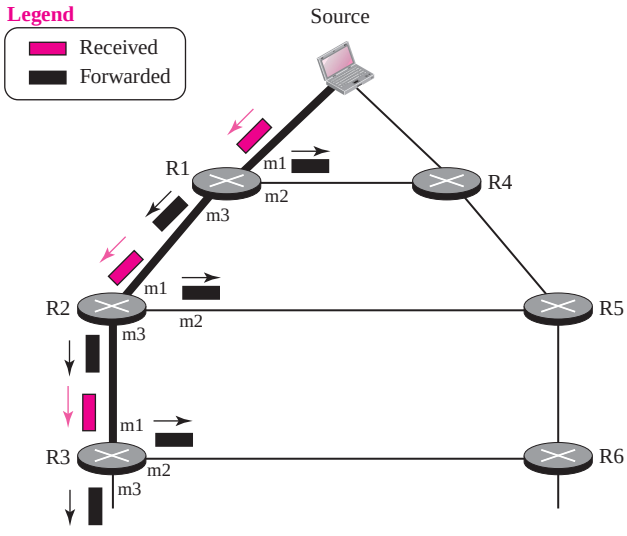
**Flooding** Flooding is the first strategy that comes to mind. A router receives a packet and without even looking at the destination group address, sends it out from every interface except the one from which it was received. **Flooding** accomplishes the first goal of multicasting: every network with active members receives the packet. However, so will networks without active members. This is a broadcast, not a multicast. There is another problem: it creates loops. A packet that has left the router may come back again from another interface or the same interface and be forwarded again. Some flooding protocols keep a copy of the packet for a while and discard any duplicates to avoid loops. The next strategy, reverse path forwarding, corrects this defect.

**Flooding broadcasts packets, but creates loops in the systems.**

**Reverse Path Forwarding (RPF)** **Reverse path forwarding (RPF)** is a modified flooding strategy. To prevent loops, only one copy is forwarded; the other copies are dropped. In RPF, a router forwards only the copy that has traveled the shortest path from the source to the router. To find this copy, RPF uses the unicast routing table. The router receives a packet and extracts the source address (a unicast address). It consults its unicast routing table as though it wants to send a packet to the source address. The routing table tells the router the next hop. If the multicast packet has just come from the hop defined in the table, the packet has traveled the shortest path from the source to the router because the shortest path is reciprocal in unicast distance vector routing protocols. If the path from A to B is the shortest, then it is also the shortest from B to A. The router forwards the packet if it has traveled from the shortest path; it discards it otherwise.

This strategy prevents loops because there is always one shortest path from the source to the router. If a packet leaves the router and comes back again, it has not traveled the shortest path. To make the point clear, let us look at Figure 12.22.

**Figure 12.22** *RPF*



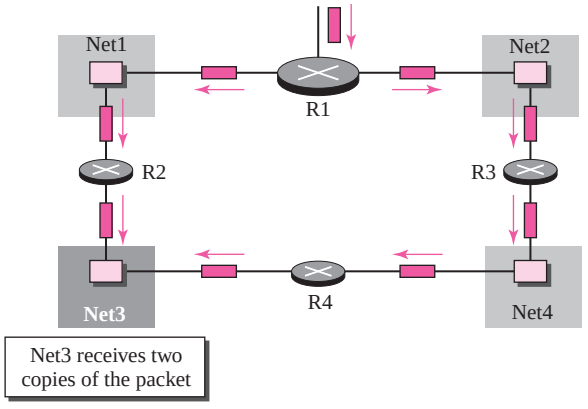

The figure shows part of a domain and a source. The shortest path tree as calculated by routers R1, R2, and R3 is shown by a thick line. When R1 receives a packet from the source through the interface m1, it consults its routing table and finds that the shortest path from R1 to the source is through interface m1. The packet is forwarded. However, if a copy of the packet has arrived through interface m2, it is discarded because m2 does not define the shortest path from R1 to the source. The story is the same with R2 and R3. You may wonder what happens if a copy of a packet that arrives at the m1 interface of R3 travels through R6, R5, R2, and then enters R3 through interface m1. This interface is the correct interface for R3. Is the copy of the packet forwarded? The answer is that this scenario never happens because when the packet goes from R5 to R2, it will be discarded by R2 and never reaches R3. The upstream routers toward the source always discard a packet that has not gone through the shortest path, thus preventing confusion for the downstream routers.

**RPF eliminates the loop in the flooding process.**

### *Reverse Path Broadcasting (RPB)*

RPF guarantees that each network receives a copy of the multicast packet without formation of loops. However, RPF does not guarantee that each network receives only one copy; a network may receive two or more copies. The reason is that RPF is not based on the destination address (a group address); forwarding is based on the source address. To visualize the problem, let us look at Figure 12.23.

**Figure 12.23** *Problem with RPF*

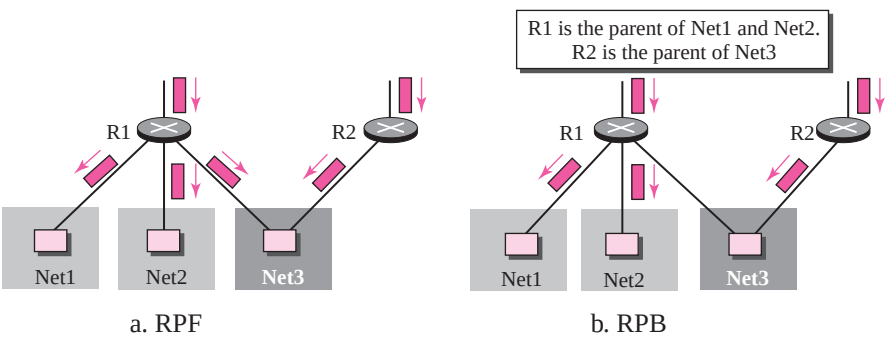



Net3 in this figure receives two copies of the packet even though each router just sends out one copy from each interface. There is duplication because a tree has not been made; instead of a tree we have a graph. Net3 has two parents: routers R2 and R4.

To eliminate duplication, we must define only one parent router for each network. We must have this restriction: A network can receive a multicast packet from a particular source only through a designated parent router.

Now the policy is clear. For each source, the router sends the packet only out of those interfaces for which it is the designated parent. This policy is called **reverse path broadcasting (RPB).** RPB guarantees that the packet reaches every network and that every network receives only one copy. Figure 12.24 shows the difference between RPF and RPB.

**Figure 12.24** *RPF versus RPB*




The reader may ask how the designated parent is determined. The designated parent router can be the router with the shortest path to the source. Because routers periodically send updating packets to each other (in RIP), they can easily determine which router in the

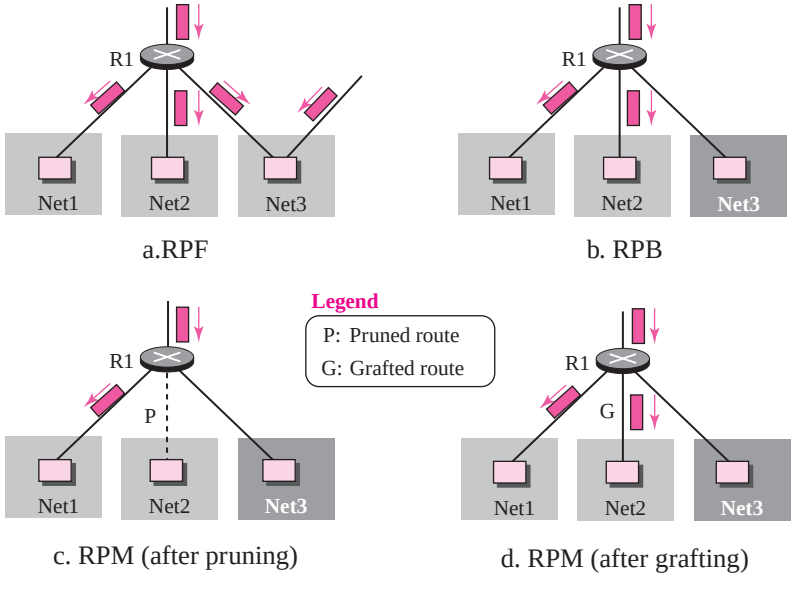
neighborhood has the shortest path to the source (when interpreting the source as the destination). If more than one router qualifies, the router with the smallest IP address is selected.

**RPB creates a shortest path broadcast tree from the source to each destination. It guarantees that each destination receives one and only one copy of the packet.**

### *Reverse Path Multicasting (RPM)*

As you may have noticed, RPB does not multicast the packet, it broadcasts it. This is not efficient. To increase efficiency, the multicast packet must reach only those networks that have active members for that particular group. This is called **reverse path multicasting (RPM).** To convert broadcasting to multicasting, the protocol uses two procedures, pruning and grafting. Figure 12.25 shows the idea of pruning and grafting.

**Figure 12.25** *RPF, RPB, and RPM*




**Pruning** The designated parent router of each network is responsible for holding the membership information. This is done through the IGMP protocol described earlier in the chapter. The process starts when a router connected to a network finds that there is no interest in a multicast packet. The router sends a **prune message** to the upstream router so that it can prune the corresponding interface. That is, the upstream router can stop sending multicast messages for this group through that interface. Now if this router receives prune messages from all downstream routers, it, in turn, sends a prune message to its upstream router.

**Grafting** What if a leaf router (a router at the bottom of the tree) has sent a prune message but suddenly realizes, through IGMP, that one of its networks is again interested in receiving the multicast packet? It can send a **graft message.** The graft message forces the upstream router to resume sending the multicast messages.

**RPM adds pruning and grafting to RPB to create a multicast shortest path tree that supports dynamic membership changes.**

## DVMRP

The **Distance Vector Multicast Routing Protocol (DVMRP)** is an implementation of multicast distance vector routing. It is a source-based routing protocol, based on RIP.

## CBT

The **Core-Based Tree (CBT) protocol** is a group-shared protocol that uses a core as the root of the tree. The autonomous system is divided into regions, and a core (center router or rendezvous router) is chosen for each region.

### *Formation of the Tree*

After the rendezvous point is selected, every router is informed of the unicast address of the selected router. Each router with an intercreated group then sends a unicast join message (similar to a grafting message) to show that it wants to join the group. This message passes through all routers that are located between the sender and the rendezvous router. Each intermediate router extracts the necessary information from the message, such as the unicast address of the sender and the interface through which the packet has arrived, and forwards the message to the next router in the path. When the rendezvous router has received all join messages from every member of the group, the tree is formed. Now every router knows its upstream router (the router that leads to the root) and the downstream router (the router that leads to the leaf).

If a router wants to leave the group, it sends a leave message to its upstream router. The upstream router removes the link to that router from the tree and forwards the message to its upstream router, and so on. Figure 12.26 shows a group-shared tree with its rendezvous router.

The reader may have noticed two differences between DVMRP and MOSPF, on one hand, and CBT, on the other. First, the tree for the first two is made from the root up; the tree for CBT is formed from the leaves down. Second, in DVMRP, the tree is first made (broadcasting) and then pruned; in CBT, there is no tree at the beginning; the joining (grafting) gradually makes the tree.

### *Sending Multicast Packets*

After formation of the tree, any source (belonging to the group or not) can send a multicast packet to all members of the group. It simply sends the packet to the rendezvous router, using the unicast address of the rendezvous router; the rendezvous router distributes the packet to all members of the group. Figure 12.27 shows how a host can send a multicast packet to all members of the group. Note that the source host can be any of

**Figure 12.26** *Group-shared tree with rendezvous router*

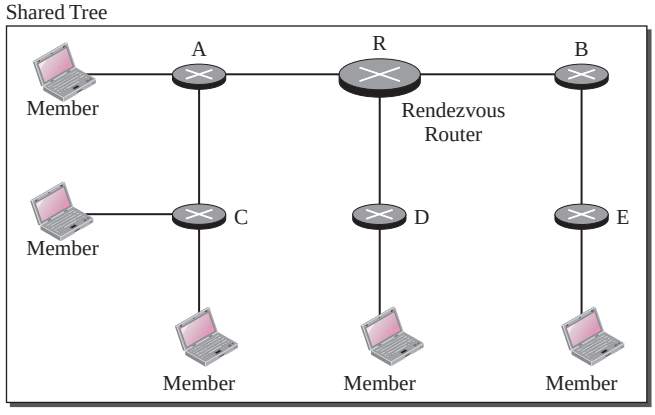
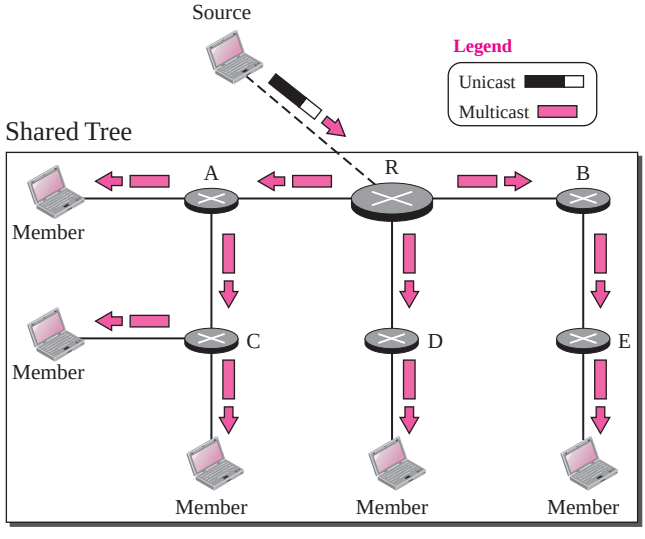



**Figure 12.27** *Sending a multicast packet to the rendezvous router*




the hosts inside the shared tree or any host outside the shared tree. In the figure we show one located outside the shared tree.

### *Selecting the Rendezvous Router*

This approach is simple except for one point. How do we select a rendezvous router to optimize the process and multicasting as well? Several methods have been implemented. However, this topic is beyond the scope of this book and we leave it to more advanced books.

### *Summary*

In summary, the Core-Based Tree (CBT) is a group-shared tree, center-based protocol using one tree per group. One of the routers in the tree is called the core. A packet is sent from the source to members of the group following this procedure:

1. The source, which may or may not be part of the tree, encapsulates the multicast packet inside a unicast packet with the unicast destination address of the core and sends it to the core. This part of delivery is done using a unicast address; the only recipient is the core router.
2. The core decapsulates the unicast packet and forwards it to all interested interfaces.
3. Each router that receives the multicast packet, in turn, forwards it to all interested interfaces.

**In CBT, the source sends the multicast packet (encapsulated in a unicast packet) to the core router. The core router decapsulates the packet and forwards it to all interested interfaces.**

## PIM

**Protocol Independent Multicast (PIM)** is the name given to two independent multicast routing protocols: **Protocol Independent Multicast, Dense Mode (PIM-DM)** and **Protocol Independent Multicast, Sparse Mode (PIM-SM)**. Both protocols are unicast-protocol dependent, but the similarity ends here. We discuss each separately.

### *PIM-DM*

PIM-DM is used when there is a possibility that each router is involved in multicasting (dense mode). In this environment, the use of a protocol that broadcasts the packet is justified because almost all routers are involved in the process.

**PIM-DM is used in a dense multicast environment, such as a LAN.**

PIM-DM is a source-based tree routing protocol that uses RPF and pruning/grafting strategies for multicasting. Its operation is like DVMRP; however, unlike DVMRP, it does not depend on a specific unicasting protocol. It assumes that the autonomous system is using a unicast protocol and each router has a table that can find the outgoing interface that has an optimal path to a destination. This unicast protocol can be a distance vector protocol (RIP) or link state protocol (OSPF).

**PIM-DM uses RPF and pruning/grafting strategies to handle multicasting. However, it is independent from the underlying unicast protocol.**

### *PIM-SM*

PIM-SM is used when there is a slight possibility that each router is involved in multicasting (sparse mode). In this environment, the use of a protocol that broadcasts the

packet is not justified; a protocol such as CBT that uses a group-shared tree is more appropriate.

**PIM-SM is used in a sparse multicast environment such as a WAN.**

PIM-SM is a group-shared tree routing protocol that has a rendezvous point (RP) as the source of the tree. Its operation is like CBT; however, it is simpler because it does not require acknowledgment from a join message. In addition, it creates a backup set of RPs for each region to cover RP failures.

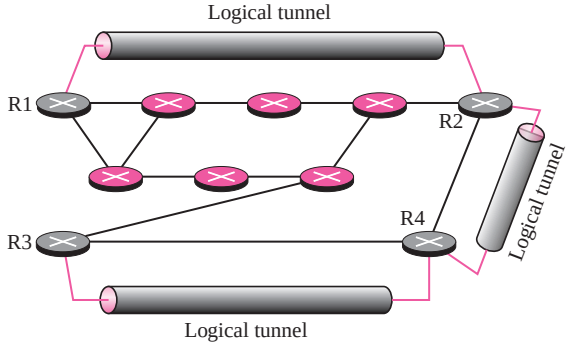
One of the characteristics of PIM-SM is that it can switch from a group-shared tree strategy to a source-based tree strategy when necessary. This can happen if there is a dense area of activity far from the RP. That area can be more efficiently handled with a source-based tree strategy instead of a group-shared tree strategy.

**PIM-SM is similar to CBT but uses a simpler procedure.**

## 12.6 MBONE

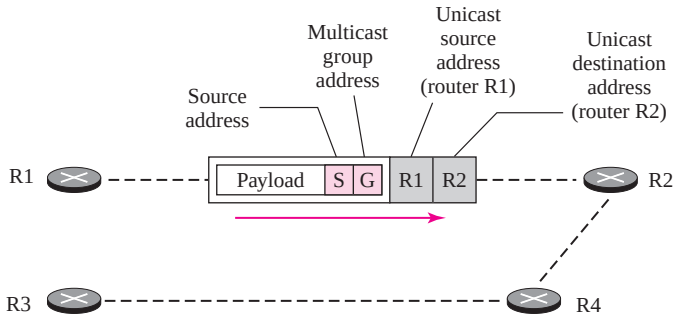
Multimedia and real-time communication have increased the need for multicasting in the Internet. However, only a small fraction of Internet routers are multicast routers. In other words, a multicast router may not find another multicast router in the neighborhood to forward the multicast packet. Although this problem may be solved in the next few years by adding more and more multicast routers, there is another solution for this problem. The solution is tunneling. The multicast routers are seen as a group of routers on top of unicast routers. The multicast routers may not be connected directly, but they are connected logically. Figure 12.28 shows the idea. In this figure, only the routers enclosed in the shaded circles are capable of multicasting. Without tunneling, these routers are isolated islands. To enable multicasting, we make a **multicast backbone (MBONE)** out of these isolated routers using the concept of tunneling.

**Figure 12.28** *Logical tunneling*



A logical tunnel is established by encapsulating the multicast packet inside a unicast packet. The multicast packet becomes the payload (data) of the unicast packet. The intermediate (non-multicast) routers forward the packet as unicast routers and deliver the packet from one island to another. It's as if the unicast routers do not exist and the two multicast routers are neighbors. Figure 12.29 shows the concept. So far the only protocol that supports MBONE and tunneling is DVMRP.

**Figure 12.29** *MBONE*




## 12.7 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], and [Wit & Zit 01].

### RFCs

Several RFCs are involved with the materials discussed in this chapter including RFC 1075, RFC 1585, RFC 2189, RFC 2362, and RFC 3376.

## 12.8 KEY TERMS

Core-Base Tree (CBT) protocol
data-driven
Distance Vector Multicast Routing Protocol (DVMRP)
flooding
graft message
group-shared tree
Internet Group Management Protocol (IGMP)

multicast backbone (MBONE)
Multicast Open Shortest Path First (MOSPF)
Protocol Independent Multicast (PIM)
Protocol Independent Multicast, Dense Mode (PIM-DM)
Protocol Independent Multicast, Sparse Mode (PIM-SM)
prune message
reverse path broadcasting (RPB)
reverse path forwarding (RPF)
reverse path multicasting (RPM)
shortest path tree
source-based tree
tunneling

## 12.9 SUMMARY

- Multicasting is the sending of the same message to more than one receiver simultaneously. Multicasting has many applications including distributed databases, information dissemination, teleconferencing, and distance learning.
- In classless addressing the block 224.0.0.0/4 is used for multicast addressing. This block is sometimes referred to as the multicast address space and is divided into several blocks (smaller blocks) for different purposes.
- The Internet Group Management Protocol (IGMP) is involved in collecting local membership group information. The last version of IGMP, IGMPv3 uses two types of messages: query and report.
- In a source-based tree approach to multicast routing, the source/group combination determines the tree. RPF, RPB, and RPM are efficient improvements to source-based trees. MOSPF/DVMRP and PIM-DM are two protocols that use source-based tree methods to multicast.
- In a group-shared approach to multicasting, one rendezvous router takes the responsibility of distributing multicast messages to their destinations. CBT and PIM-SM are examples of group-shared tree protocols.
- For multicasting between two noncontiguous multicast routers, we make a multicast backbone (MBONE) to enable tunneling.

## 12.10 PRACTICE SET

### Exercises

1. Exactly describe why we cannot use the CIDR notation for the following blocks in Table 12.1:
   a. AD HOC block with the range 224.0.2.0 to 224.0.255.255.
   b. The first reserved block with the range 224.3.0.0 to 231.255.255.255.
   c. The second reserved block with the range 234.0.0.0 to 238.255.255.255
2. The AS number in an organization is 24101. Find the range of multicast addresses that the organization can use in the GLOP block.
3. A multicast address for a group is 232.24.60.9. What is its 48-bit Ethernet address for a LAN using TCP/IP?

4. Change the following IP multicast addresses to Ethernet multicast addresses. How many of them specify the same Ethernet address?
   a. 224.18.72.8
   b. 235.18.72.8
   c. 237.18.6.88
   d. 224.88.12.8
5. Why is there no need for the IGMP message to travel outside its own network?
6. Answer the following questions:
   a. What is the size of a general query message in IGMPv3?
   b. What is the size of a group-specific message in IGMPv3?
   c. What is the size of a group-and-source-specific message in IGMPv3 if it contains 10 source addresses?
7. What is the size of a report message in IGMPv3 if it contains 3 records and each record contain 5 source addresses (ignore auxiliary data)?
8. Show the socket state table (similar to Figure 12.11) for a host with two sockets: S1 and S2. S1 is the member of group 232.14.20.54 and S2 is the member of the group 232.17.2.8. S1 likes to receive multicast messages only from 17.8.5.2; S2 likes to receive multicast messages from all sources except 130.2.4.6.
9. Show the interface state for the computer in Exercise 8.
10. Show the group record sent by the host in Exercise 9 if socket S1 declares that it likes also to receive messages from the source 24.8.12.6.
11. Show the group record sent by the host in Exercise 9 if socket S2 declares that it wants to leave the group 232.17.2.8.
12. Show the group record sent by the host in Exercise 9 if socket S1 declares that it wants to join the group 232.33.33.7 and accept any message from any source.
13. In Figure 12.14, show the state for interface m1 if the router receives a report with the record telling a host wants to join the group 232.77.67.60 and accept the messages from any source.
14. Show the value of MRT in second if the Max Response Code is:
   a. 125
   b. 220
15. The contents of an IGMP message in hexadecimal notation are:

**11 03 EE FF E8 0E 15 08**

   Answer the following questions:
   a. What is the type?
   b. What is the checksum?
   c. What is the groupid?
16. The contents of an IGMP message in hexadecimal notation are:

**22 00 F9 C0 00 00 00 02**

Answer the following questions:

**a.** What is the type?

**b.** What is the checksum?

**c.** What is the number of records?

**17.** Change the following IP multicast addresses to Ethernet multicast addresses. How many of them specify the same Ethernet address?

**a.** 224.18.72.8

**b.** 235.18.72.8

**c.** 237.18.6.88

**d.** 224.88.12.8

**18.** In Figure 12.18, find the unicast routing tables for routers R2, R3, and R4. Show the shortest path trees.

**19.** In Figure 12.19, find the multicast routing tables for routers R2, R3, and R4.

**20.** A router using DVMRP receives a packet with source address 10.14.17.2 from interface 2. If the router forwards the packet, what are the contents of the entry related to this address in the unicast routing table?

**21.** Router A sends a unicast RIP update packet to router B that says 134.23.0.0/16 is 7 hops away. Network B sends an update packet to router A that says 13.23.0.0/16 is 4 hops away. If these two routers are connected to the same network, which one is the designated parent router?

**22.** Does RPF actually create a shortest path tree? Explain.

**23.** Does RPB actually create a shortest path tree? Explain. What are the leaves of the tree?

**24.** Does RPM actually create a shortest path tree? Explain. What are the leaves of the tree?

## Research Activities

**25.** Use *netstat* to find if your server supports multicast addressing.

**26.** Find the format of the DVMRP prune message. What is the format of the graft message?

**27.** For MOSPF find the format of the group-membership-LSA packet that associates a network with a group.

**28.** CBT uses nine types of packets. Use the Internet to find the purpose and format of each packet.

**29.** Use the Internet to find how CBT messages are encapsulated.

**30.** Use the Internet to find information regarding the scalability of each multicast routing protocol we discussed. Make a table and compare them.

# PART 3

# Transport Layer

CHAPTER

# 13 Introduction to the Transport Layer

This chapter discusses general services that a transport-layer protocol can provide and the issues related to these services. The chapter also describes the behavior of some generic transport-layer protocols designed in response to different situations. We will see, in the next chapters, how these generic protocols are combined to create transport layer protocols in the TCP/IP protocol suite such as UDP, TCP, and SCTP.

## OBJECTIVE

*We have several objectives for this chapter:*

- To define process-to-process communication at the transport layer and compare it with host-to-host communication at the network layer.
- To discuss the addressing mechanism at the transport layer, to discuss port numbers, and to define the range port numbers used for different purposes.
- To explain the packetizing issue at the transport layer: encapsulation and decapsulation of messages.
- To discuss multiplexing (many-to-one) and demultiplexing (one-to-many) services provided by the transport layer.
- To discuss flow control and how it can be achieved at the transport layer.
- To discuss error control and how it can be achieved at the transport layer.
- To discuss congestion control and how it can be achieved at the transport layer.
- To discuss the connectionless and connection-oriented services at the transport layer and show their implementation using an FSM.
- To discuss the behavior of four generic transport-layer protocols and their applications: simple protocol, Stop-and-Wait protocol, Go-Back-*N* protocol, and Selective-Repeat protocol.
- To describe the idea of bidirectional communication at the transport layer using the piggybacking method.

## 13.1 TRANSPORT-LAYER SERVICES

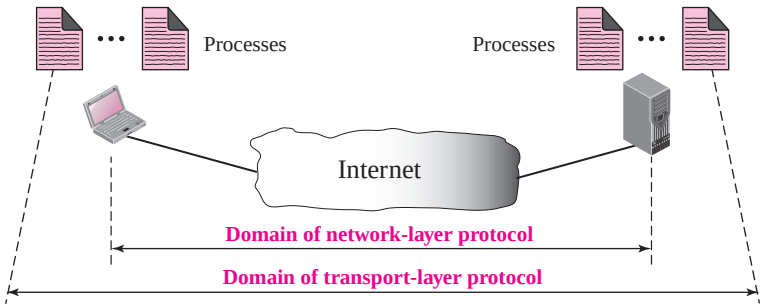
As we discussed in Chapter 2, the transport layer is located between the network layer and the application layer. The transport layer is responsible for providing services to the application layer; it receives services from the network layer. In this section, we discuss the services that can be provided by a transport layer; in the next section, we discuss the principle beyond several transport layer protocols.

### Process-to-Process Communication

The first duty of a transport-layer protocol is to provide **process-to-process communication.** A process is an application-layer entity (running program) that uses the services of the transport layer. Before we discuss how process-to-process communication can be accomplished, we need to understand the difference between host-to-host communication and process-to-process communication.

The network layer is responsible for communication at the computer level (host-to-host communication). A network layer protocol can deliver the message only to the destination computer. However, this is an incomplete delivery. The message still needs to be handed to the correct process. This is where a transport layer protocol takes over. A transport layer protocol is responsible for delivery of the message to the appropriate process. Figure 13.1 shows the domains of a network layer and a transport layer.

**Figure 13.1** *Network layer versus transport layer*




### Addressing: Port Numbers

Although there are a few ways to achieve process-to-process communication, the most common is through the **client-server paradigm** (see Chapter 17). A process on the local host, called a *client,* needs services from a process usually on the remote host, called a *server.*

Both processes (client and server) have the same name. For example, to get the day and time from a remote machine, we need a daytime client process running on the local host and a daytime server process running on a remote machine.

However, operating systems today support both multiuser and multiprogramming environments. A remote computer can run several server programs at the same time, just as several local computers can run one or more client programs at the same time. For communication, we must define the

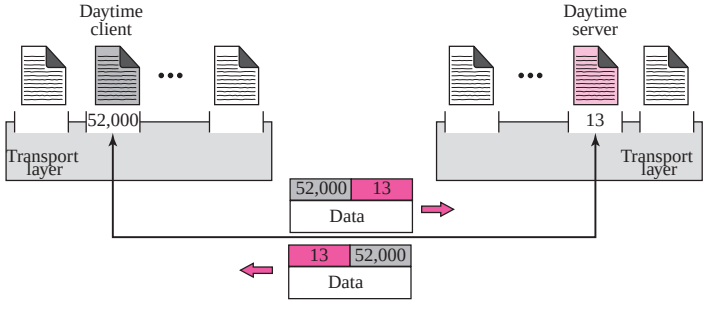
- Local host
- Local process
- Remote host
- Remote process

The local host and the remote host are defined using IP addresses. To define the processes, we need second identifiers called **port numbers.** In the TCP/IP protocol suite, the port numbers are integers between 0 and 65,535.

The client program defines itself with a port number, called the **ephemeral port number.** The word ephemeral means *short lived* and is used because the life of a client is normally short. An ephemeral port number is recommended to be greater than 1,023 for some client/server programs to work properly.

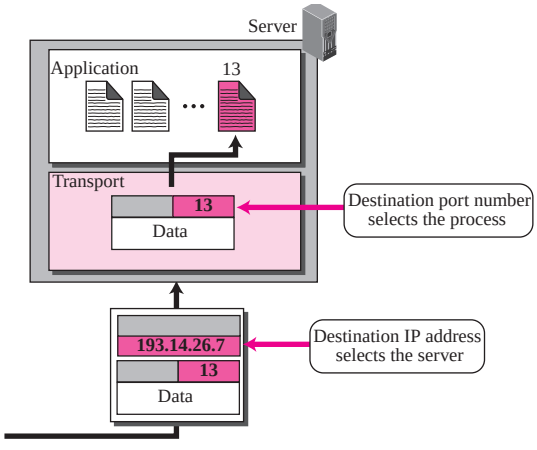
The server process must also define itself with a port number. This port number, however, cannot be chosen randomly. If the computer at the server site runs a server process and assigns a random number as the port number, the process at the client site that wants to access that server and use its services will not know the port number. Of course, one solution would be to send a special packet and request the port number of a specific server, but this creates more overhead. TCP/IP has decided to use universal port numbers for servers; these are called **well-known port numbers.** There are some exceptions to this rule; for example, there are clients that are assigned well-known port numbers. Every client process knows the well-known port number of the corresponding server process. For example, while the daytime client process, discussed above, can use an ephemeral (temporary) port number 52,000 to identify itself, the daytime server process must use the well-known (permanent) port number 13. Figure 13.2 shows this concept.

**Figure 13.2** *Port numbers*



It should be clear by now that the IP addresses and port numbers play different roles in selecting the final destination of data. The destination IP address defines the host among the different hosts in the world. After the host has been selected, the port number defines one of the processes on this particular host (see Figure 13.3).

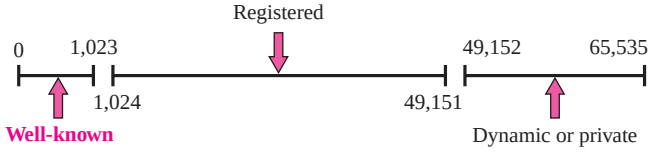
**Figure 13.3** *IP addresses versus port numbers*




### *ICANN Ranges*

ICANN has divided the port numbers into three ranges: well-known, registered, and dynamic (or private), as shown in Figure 13.4.

**Figure 13.4** *ICANN ranges*




- **Well-known ports.** The ports ranging from 0 to 1,023 are assigned and controlled by ICANN. These are the well-known ports.
- **Registered ports.** The ports ranging from 1,024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.
- **Dynamic ports.** The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used as temporary or private port numbers. The original recommendation was that the ephemeral port numbers for clients be chosen from this range. However, most systems do not follow this recommendation.

**The well-known port numbers are less than 1,024.**

### Example 13.1

In UNIX, the well-known ports are stored in a file called /etc/services. Each line in this file gives the name of the server and the well-known port number. We can use the grep utility to extract the line corresponding to the desired application. The following shows the port for TFTP. Note that TFTP can use port 69 on either UDP or TCP. SNMP (see Chapter 24) uses two port numbers (161 and 162), each for a different purpose.

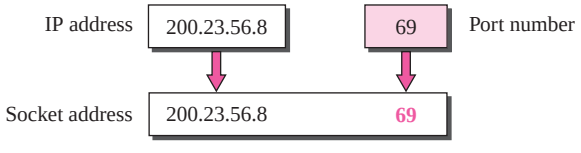
```
$grep    tftp   /etc/services
   tftp          69/tcp
   tftp          69/udp
```

```
$grep    snmp /etc/services
   snmp161/tcp#Simple Net Mgmt Proto
   snmp161/udp#Simple Net Mgmt Proto
   snmptrap162/udp#Traps for SNMP
```

### *Socket Addresses*

A transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end, to make a connection. The combination of an IP address and a port number is called a **socket address.** The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely (see Figure 13.5).

**Figure 13.5** *Socket address*



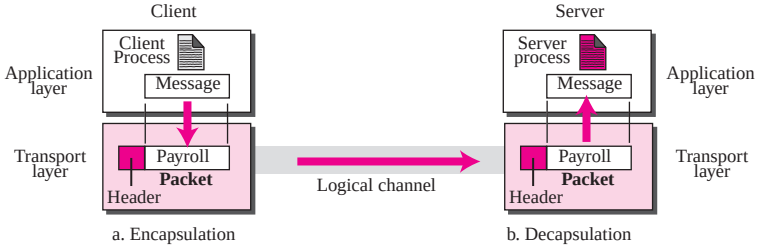

To use the services of transport layer in the Internet, we need a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the network-layer packet header and the transport-layer packet header. The first header contains the IP addresses; the second header contains the port numbers.

## Encapsulation and Decapsulation

To send a message from one process to another, the transport layer protocol encapsulates and decapsulates messages (Figure 13.6).

Encapsulation happens at the sender site. When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and

**Figure 13.6** *Encapsulation and decapsulation*




some other pieces of information that depends on the transport layer protocol. The transport layer receives the data and adds the transport-layer header. The packets at the transport layers in the Internet are called *user datagrams*, *segments*, or *packets*. We call them packets in this chapter.

Decapsulation happens at the receiver site. When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer. The sender socket address is passed to the process in case it needs to respond to the message received.

## Multiplexing and Demultiplexing

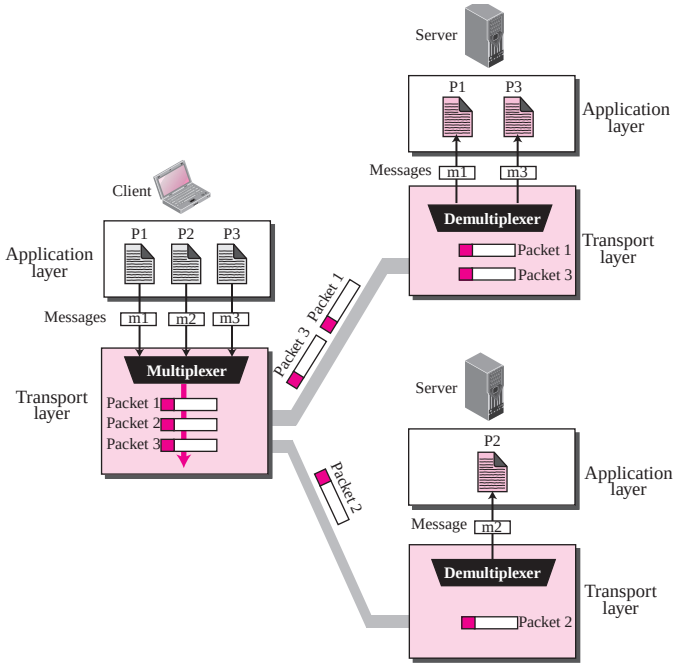
Whenever an entity accepts items from more than one source, it is referred to as **multiplexing** (many to one); whenever an entity delivers items to more than one source, it is referred to as **demultiplexing** (one to many). The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing (Figure 13.7).

Figure 13.7 shows communication between a client and two servers. Three client processes are running at the client site, P1, P2, and P3. The processes P1 and P3 need to send requests to the corresponding server process running in a server. The client process P2 needs to send a request to the corresponding server process running at another server. The transport layer at the client site accepts three messages from the three processes and creates three packets. It acts as a *multiplexer*. The packets 1 and 3 use the same logical channel to reach the transport layer of the first server. When they arrive at the server, the transport layer does the job of a *multiplexer* and distributes the messages to two different processes. The transport layer at the second server receives packet 2 and delivers it to the corresponding process.

## Flow Control

Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates. If the items are produced faster than they can be consumed, the consumer can be overwhelmed and needs to discard some items. If the items are produced slower than they can be consumed, the consumer should wait; the system becomes less efficient. Flow control is related to the first issue. We need to prevent losing the data items at the consumer site.

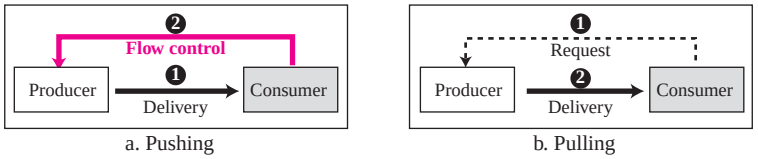
**Figure 13.7** *Multiplexing and demultiplexing*




## *Pushing or Pulling*

Delivery of items from a producer to a consumer can occur in one of the two ways: *pushing* or *pulling*. If the sender delivers items whenever they are produced—without the prior request from the consumer—the delivery is referred to as pushing. If the producer delivers the items after the consumer has requested them, the delivery is referred to as pulling. Figure 13.8 shows these two types of delivery.

**Figure 13.8** *Pushing or pulling*



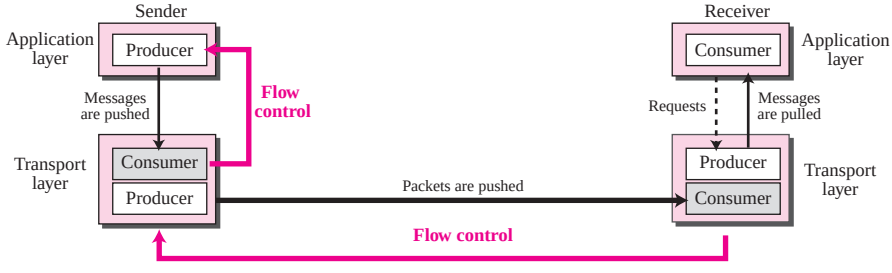

When the producer *pushes* the items, the consumer may be overwhelmed and there is a need for flow control, in the opposite direction, to prevent the discarding of the items. In other words, the consumer needs to warn the producer to stop the delivery and to inform it when it is ready again to receive the items. When the consumer pulls the items, it requests them when it is ready. In this case, there is no need for flow control.

### Flow Control at Transport Layer

In communication at the transport layer, we are dealing with four entities: sender process, sender transport layer, receiver transport layer, and receiver process. The sending process at the application layer is only a producer. It produces message chunks and pushes them to the transport layer. The sending transport layer has a double role: it is both a consumer and the producer. It consumes the messages pushed by the producer. It encapsulates the messages in packets and pushes them to the receiving transport layer. The receiving transport layer has also a double role: it is the consumer for the packets received from the sender. It is also a producer; it needs to decapsulate the messages and deliver them to the application layer. The last delivery, however, is normally a pulling delivery; the transport layer waits until the application-layer process asks for messages.

Figure 13.9 shows that we need at least two cases of flow control: from the sending transport layer to the sending application layer and from the receiving transport layer to the sending transport layer.

**Figure 13.9** *Flow control at the transport layer*




### Buffers

Although flow control can be implemented in several ways, one of the solutions is normally to use two *buffers*. One at the sending transport layer and the other at the receiving transport layer. A buffer is a set of memory locations that can hold packets at the sender and receiver. The flow control communication can occur by sending signals from the consumer to producer.

When the buffer of the sending transport layer is full, it informs the application layer to stop passing chunks of messages; when there are some vacancies, it informs the application layer that it can pass message chunks again.

When the buffer of the receiving transport layer is full, it informs the sending transport layer to stop sending packets. When there are some vacancies, it informs the sending transport layer that it can send message again.

### Example 13.2

The above discussion requires that the consumers communicate with the producers in two occasions: when the buffer is full and when there are vacancies. If the two parties use a buffer of only one slot, the communication can be easier. Assume that each transport layer uses one single

memory location to hold a packet. When this single slot in the sending transport layer is empty, the sending transport layer sends a note to the application layer to send its next chunk; when this single slot in the receiving transport layer is empty, it sends an acknowledgment to the sending transport layer to send its next packet. As we will see later, this type of flow control, using a single-slot buffer at the sender and the receiver, is inefficient.

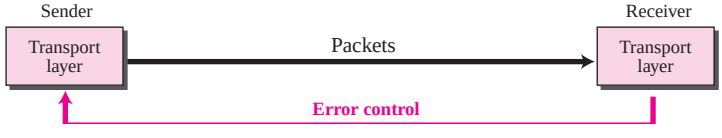
## Error Control

In the Internet, since the underlying network layer (IP), which is responsible to carry the packets from the sending transport layer to the receiving transport layer, is unreliable, we need to make the transport layer reliable if the application requires reliability. Reliability can be achieved to add error control service to the transport layer. Error control at the transport layer is responsible to

1. Detect and discard corrupted packets.
2. Keep track of lost and discarded packets and resend them.
3. Recognize duplicate packets and discard them.
4. Buffer out-of-order packets until the missing packets arrive.

Error control, unlike the flow control, involves only the sending and receiving transport layers. We are assuming that the message chunks exchanged between the application and transport layers are error free. Figure 13.10 shows the error control between the sending and receiving transport layer. As with the case of flow control, the receiving transport layer manages error control, most of the time, by informing the sending transport layer about the problems.

**Figure 13.10** *Error control at the transport layer*




### *Sequence Numbers*

Error control requires that the sending transport layer knows which packet is to be resent and the receiving transport layer knows which packet is a duplicate, or which packet has arrived out of order. This can be done if the packets are numbered. We can add a field to the transport layer packet to hold the **sequence number** of the packets. When a packet is corrupted or lost, the receiving transport layer can somehow inform the sending transport layer to resend that packet using the sequence number. The receiving transport layer can also detect duplicate packets if two received packets have the same sequence number. The out-of-order packets can be recognized by observing gaps in the sequence numbers.

Packets are numbered sequentially. However, because we need to include the sequence number of each packet in the header, we need to set a limit. If the header of

the packet allows *m* bits for the sequence number, the sequence numbers range from 0 to $2^m - 1$. For example, if *m* is 4, the only sequence numbers are 0 through 15, inclusive. However, we can wrap around the sequence. So the sequence numbers in this case are

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, …

In other words, the sequence numbers are modulo $2^m$.

**For error control, the sequence numbers are modulo $2^m$, where *m* is the size of the sequence number field in bits.**

### *Acknowledgment*

We can use both positive and negative signals as error control. The receiver side can send an acknowledgement (ACK) for each or a collection of packets that have arrived safe and sound. The receiver can simply discard the corrupted packets. The sender can detect lost packets if it uses a timer. When a packet is sent, the sender starts a timer; when the timer expires, if an ACK does not arrive before the timer expires, the sender resends the packet. Duplicate packets can be silently discarded by the receiver. Out-of-order packets can be either discarded (to be treated as lost packets by the sender), or stored until the missing ones arrives.

## Combination of Flow and Error Control

We have discussed that flow control requires the use of two buffers, one at the sender site and the other at the receiver site. We have also discussed that the error control requires the use of sequence and acknowledgment numbers by both sides. These two requirements can be combined if we use two numbered buffers, one at the sender, one at the receiver.

At the sender, when a packet is prepared to be sent, we use the number of the next free location, *x*, in the buffer as the sequence number of the packet. When the packet is sent, a copy is stored at memory location *x*, awaiting the acknowledgment from the other end. When an acknowledgment related to a sent packet arrives, the packet is purged and the memory location becomes free.

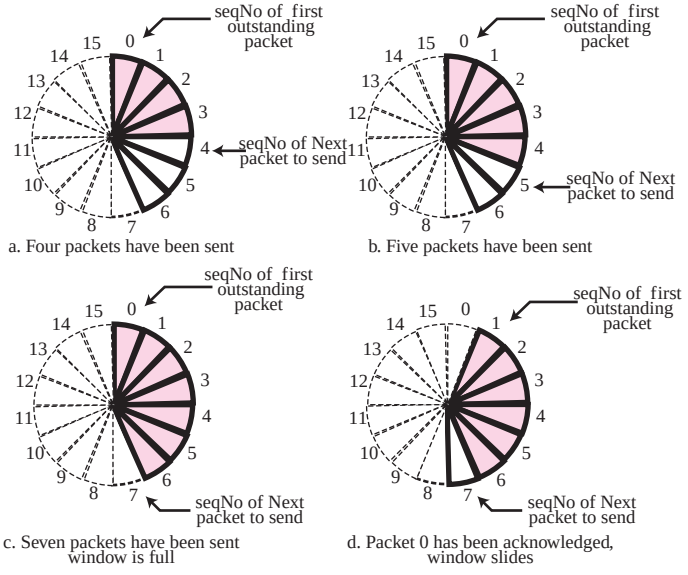
At the receiver, when a packet with sequence number y arrives, it is stored at the memory location *y* until the application layer is ready to receive it. An acknowledgment can be sent to announce the arrival of packet *y*.

### *Sliding Window*

Since the sequence numbers used modulo $2^m$, a circle can represent the sequence number from 0 to $2^m - 1$ (Figure 13.11).

The buffer is represented as a set of slices, called the **sliding window,** that occupy part of the circle at any time. At the sender site, when a packet is sent, the corresponding slice is marked. When all the slices are marked, it means that the buffer is full and no further messages can be accepted from the application layer. When an acknowledgment

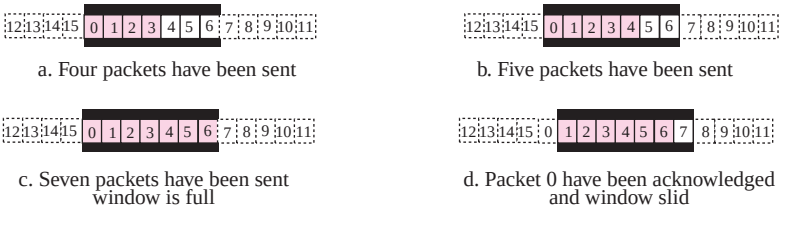
**Figure 13.11** *Sliding window in circular format*




arrives, the corresponding slice is unmarked. If some consecutive slices from the beginning of the window are unmarked, the window slides over the range of the corresponding sequence number to allow more free slices at the end of the window. Figure 13.11 shows the sliding window at the sender. The sequence number are modulo 16 ($m$ = 4) and the size of the window is 7. Note that the sliding window is just an abstraction: the actual situation uses computer variables to hold the sequence number of the next packet to be sent and the last packet sent.

Most protocols show the sliding window using linear representation. The idea is the same, but it normally takes less space on paper. Figure 13.12 shows this representation. Both representations tell us the same thing. If we take both sides of each part in Figure 13.11 and bend them up, we can make the same part in Figure 13.12.

**Figure 13.12** *Sliding window in linear format*



## Congestion Control

An important issue in the Internet is **congestion.** Congestion in a network may occur if the **load** on the network—the number of packets sent to the network—is greater than the *capacity* of the network—the number of packets a network can handle. **Congestion control** refers to the mechanisms and techniques to control the congestion and keep the load below the capacity.

We may ask why there is congestion on a network. Congestion happens in any system that involves waiting. For example, congestion happens on a freeway because any abnormality in the flow, such as an accident during rush hour, creates blockage.

Congestion in a network or internetwork occurs because routers and switches have queues—buffers that hold the packets before and after processing. The packet is put in the appropriate output queue and waits its turn to be sent. These queues are finite, so it is possible for more packets to arrive at a router than the router can buffer.

Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened.

### *Open-Loop Congestion Control*

In **open-loop congestion control,** policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination.

**Retransmission Policy** Retransmission is sometimes unavoidable. If the sender feels that a sent packet is lost or corrupted, the packet needs to be retransmitted. Retransmission in general may increase congestion in the network. However, a good retransmission policy can prevent congestion. The retransmission policy and the retransmission timers must be designed to optimize efficiency and at the same time prevent congestion.

**Window Policy** The type of window at the sender may also affect congestion. We will see later in the chapter that the *Selective Repeat* window is better than the *Go-Back-N* window for congestion control.

**Acknowledgment Policy** The acknowledgment policy imposed by the receiver may also affect congestion. If the receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion. Several approaches are used in this case. A receiver may send an acknowledgment only if it has a packet to be sent or a special timer expires. A receiver may decide to acknowledge only *N* packets at a time. We need to know that the acknowledgments are also part of the load in a network. Sending fewer acknowledgments means imposing less load on the network.

### *Closed-Loop Congestion Control*

**Closed-loop congestion control** mechanisms try to alleviate congestion after it happens. Several mechanisms have been used by different protocols. We describe the one used in the transport layer. The size of the window at the sender size can be flexible. One factor that can determine the sender window size is the congestion in the Internet. The sending transport layer can monitor the congestion in the Internet, by watching the lost packets, and use a strategy to decrease the window size if the congestion is increasing and vice versa. We see in Chapter 15 how TCP uses this strategy to control its window size.

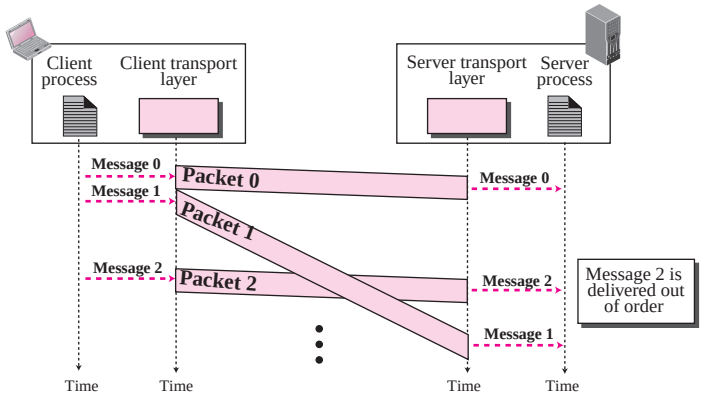
## Connectionless and Connection-Oriented Services

A transport-layer protocol, like a network-layer protocol can provide two types of services: connectionless and connection-oriented. The nature of these services at the transport layer, however, is different from the ones at the network layer. At the network layer, a connectionless service may mean different paths for different datagrams belonging to the same message. At the transport layer, we are not concerned about the physical paths of packets (we assume a logical connection between two transport layers), connectionless service at the transport layer means independency between packets; connection-oriented means dependency. Let us elaborate on these two services.

### *Connectionless Service*

In a connectionless service, the source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one. The transport layer treats each chunk as a single unit without any relation between the chunks. When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it. To show the independency of packets, assume that a client process has three chunks of messages to send a server process. The chunks are handed over to the connectionless transport protocol in order. However, since there is no dependency between the packets at the transport layer, the packets may arrive out of order at the destination and will be delivered out of order to the server process. In Figure 13.13, we have shown the movement of packets using a time line, but we have assumed that the delivery of the process to the transport layer and vice versa are instantaneous.

**Figure 13.13** *Connectionless service*




The figure shows that at the client site, the three chunks of messages are delivered to the client transport layer in order (1, 2, and 3). Because of the extra delay in transportation of the second packet, the delivery of messages at the server is not in order (1, 3, 2). If these three chunks of data belong to the same message, the server process may have received a strange message.

The situation would be worse if one of the packets were lost. Since there is no numbering on the packets, the receiving transport layer has no idea that one of the messages has been lost. It just delivers two chunks of data to the server process.

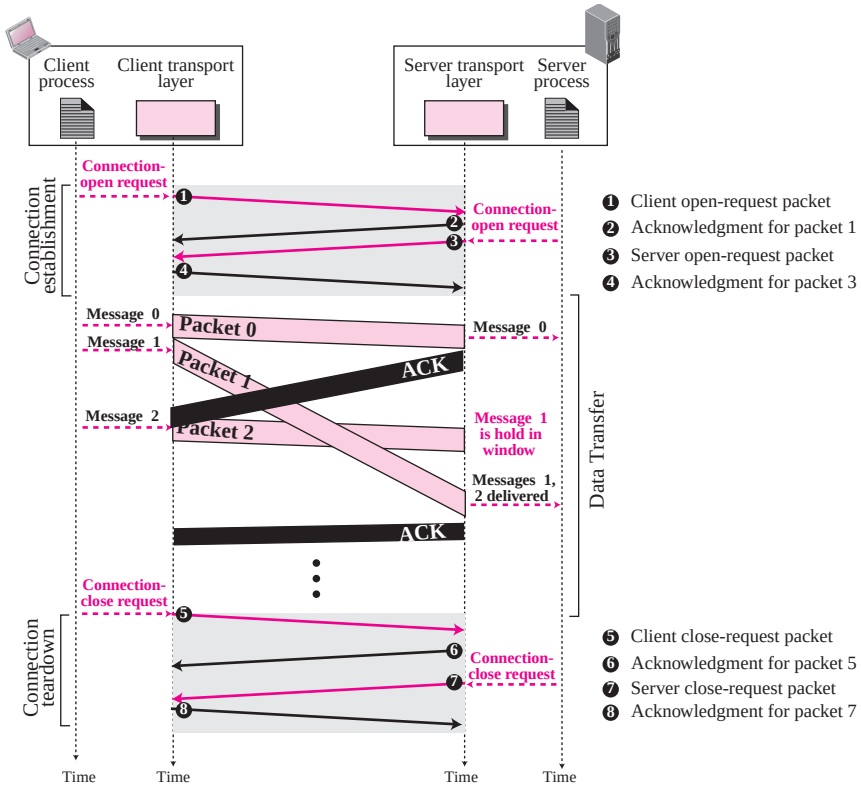
The above two problems arise from the fact that the two transport layers do not coordinate with each other. The receiving transport layer does not know when the first packet will come nor when all of the packets have arrived.

We can say that no flow control, error control, or congestion control can be effectively implemented in a connectionless service.

### *Connection-Oriented Service*

In a connection-oriented service, the client and the server first need to establish a connection between themselves.The data exchange can only happen after the connection establishment. After data exchange, the connection needs to be teared down (Figure 13.14). As we mentioned before, the connection-oriented service at the transport layer is different from the same service at the network layer. In the network layer, connection-oriented

**Figure 13.14** *Connection-oriented service*



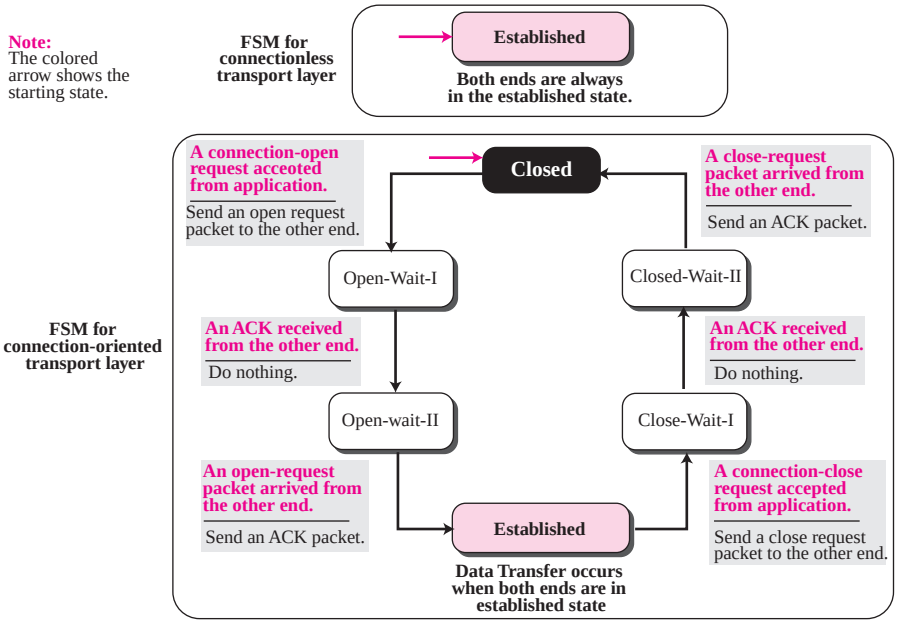
service means a coordination between the two end hosts and all the routers in between. At the transport layer, connection-oriented service involves only the two hosts; the service is end to end. This means that we should be able to make a connection-oriented protocol over either a connectionless or connection-oriented protocol. Figure 13.14 shows the connection establishment, data transfer, and teardown phases in a connection-oriented service at the transport layer. Note that most protocols combine the third and fourth packets in the connection establishment phase into one packet, as we will see in Chapters 15 and 16.

We can implement flow control, error control, and congestion control in a connection-oriented protocol.

### *Finite State Machine*

The behavior of a transport layer protocol, both when it provides a connectionless and when it provides a connection-oriented protocol, can be better shown as a **finite state machine (FSM).** Figure 13.15 shows the representation of a transport layer using an FSM. Using this tool, each transport layer (sender or receiver) is taught as a machine with a finite number of states. The machine is always in one of the states until an *event* occurs. Each event is associated with two reactions: defining the list (possibly empty) of actions to be performed and determining the next state (which can be the same as the current state). One of the states must be defined as the initial state, the state in which the machine starts when it turns on. In this figure we have used ovals to show states,

**Figure 13.15** *Connectionless and connection-oriented service represented as FSMs*



color text to show events, and regular black text to show actions. The initial state has an incoming arrow from another state. A horizontal line is used to separate the event from the actions, although in Chapter 15 we replace the horizontal line with a slash. The arrow shows the movement to the next state.

We can think of a connectionless transport layer as an FSM with only one single state: the established state. The machine on each end (client and server) is always in the established state, ready to send and receive transport-layer packets.

An FSM in a connection-oriented transport layer, on the other hand, needs to go through three states before reaching the established state. The machine also needs to go through three states before closing the connection. The machine is in the *closed* state when there is no connection. It remains in this state until a request for opening the connection arrives from the local process; the machine sends an open request packet to the remote transport layer and moves to the *open-wait-I* state. When an acknowledgment is received from the other end, the local FSM moves to the *open-wait-II* state. When the machine is in this state, a unidirectional connection has been established, but if bidirectional connection is needed, the machine needs to wait in this state until the other end also requests a connection. When the request is received, the machine sends an acknowledgment and moves to the *established* state.

Data and data acknowledgment can be exchanged between the two ends when they are both in the established state. However, we need to remember that the established state, both in connectionless and connection-oriented transport layers, represents a set of data transfer states that we discuss in the next section, Transport-Layer Protocols.

To tear down a connection, the application layer sends a close request message to its local transport layer. The transport layer sends a close-request packet to the other end and moves to *close-wait-I* state. When an acknowledgment is received from the other end, the machine moves to the *close-wait-II* state and waits for the close-request packet from the other end. When this packet arrives, the machine sends an acknowledgment and moves to the *closed* state.

There are several variations of the connection-oriented FSM that we will discuss in Chapters 15 and 16. In Chapters 15 and 16, we also see how the FSM can be condensed or expanded and the names of the states can be changed.

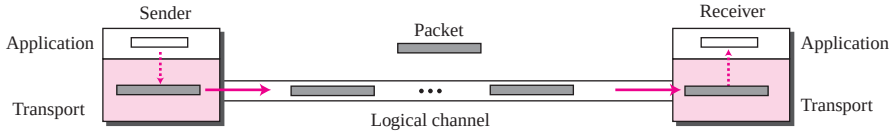
## 13.2 TRANSPORT-LAYER PROTOCOLS

We can create a transport-layer protocol by combining a set of services described in the previous sections. To better understand the behavior of these protocols, we start with the simplest one and gradually add more complexity. The TCP/IP protocol uses a transport layer protocol that is either a modification or a combination of some of these protocols. This is the reason that we discuss these general protocols in this chapter to pave the way for understanding more complex ones in the next three chapters. To make our discussion simpler, we first discuss all of these protocols as a unidirectional protocol (i.e., simplex) in which the data packets move in one direction. At the end of the chapter, we briefly discuss how they can be changed to bidirectional protocols where data can be moved in two directions (i.e., full duplex).

## Simple Protocol

Our first protocol is a simple connectionless protocol with neither flow nor error control. We assume that the receiver can immediately handle any packet it receives. In other words, the receiver can never be overwhelmed with incoming packets. Figure 13.16 shows the layout for this protocol.

**Figure 13.16** *Simple protocol*

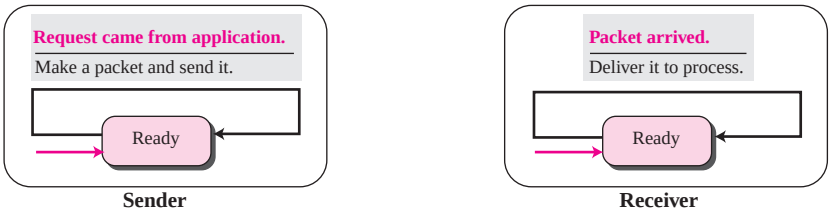



The transport layer at the sender gets a message from its application layer, makes a packet out of it, and sends the packet. The transport layer at the receiver receives a packet from its network layer, extracts the message from the packet, and delivers the message to its application layer. The transport layers of the sender and receiver provide transmission services for their application layers.

### *FSMs*

The sender site should not send a packet until its application layer has a message to send. The receiver site cannot deliver a message to its application layer until a packet arrives. We can show these requirements using two FSMs. Each FSM has only one state, the *ready state*. The sending machine remains in the ready state until a request comes from the process in the application layer. When this event occurs, the sending machine encapsulates the message in a packet and sends it to the receiving machine. The receiving machine remains in the ready state until a packet arrives from the sending machine. When this event occurs, the receiving machine decapsulates the message out of the packet and delivers it to the process at the application layer. Figure 13.17 shows the FSMs for the simple protocol. We see in Chapter 14 that UDP protocol is a slight modification of this protocol.

**Figure 13.17** *FSMs for the simple protocol*

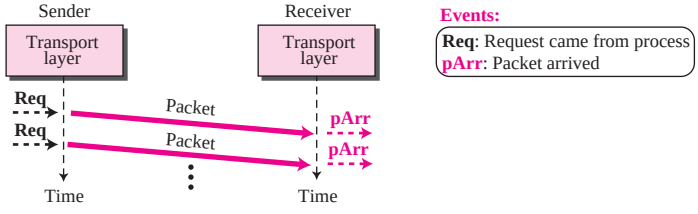



**The simple protocol is a connectionless protocol that provides neither flow nor error control.**

### Example 13.3

Figure 13.18 shows an example of communication using this protocol. It is very simple. The sender sends packets one after another without even thinking about the receiver.

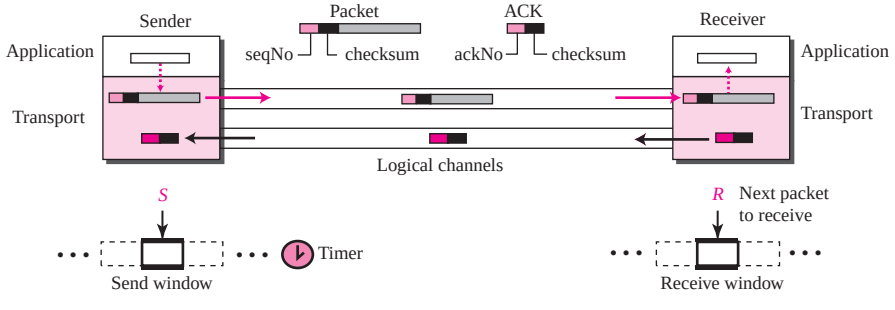
**Figure 13.18** *Flow diagram for Example 13.3*




## Stop-and-Wait Protocol

Our second protocol is a connection-oriented protocol called the **Stop-and-Wait protocol,** which uses both flow and error control. Both the sender and the receiver use a sliding window of size 1. The sender sends one packet at a time and waits for an acknowledgment before sending the next one.To detect corrupted packets, we need to add a checksum to each data packet. When a packet arrives at the receiver site, it is checked. If its checksum is incorrect, the packet is corrupted and silently discarded. The silence of the receiver is a signal for the sender that a packet was either corrupted or lost. Every time the sender sends a packet, it starts a timer. If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next packet (if it has one to send). If the timer expires, the sender resends the previous packet assuming that either the packet was lost or corrupted. This means that the sender needs to keep a copy of the packet until its acknowledgment arrives. Figure 13.19 shows the outline for the Stop-and-Wait protocol. Note that only one packet and one acknowledgment can be in the channels at any time.

**Figure 13.19** *Stop-and-Wait protocol*




The Stop-and-Wait protocol is a connection-oriented protocol that provides flow and error control.

**In Stop-and-Wait protocol, flow control is achieved by forcing the sender to wait for an acknowledgment, and error control is achieved by discarding corrupted packets and letting the sender resend unacknowledged packets when the timer expires.**

### *Sequence Numbers*

To prevent duplicate packets, the protocol uses sequence numbers and acknowledgment numbers. A field is added to the packet header to hold the sequence number of that packet. One important consideration is the range of the sequence numbers. Since we want to minimize the packet size, we look for the smallest range that provides unambiguous communication. Let us reason out the range of sequence numbers we need. Assume we have used $x$ as a sequence number; we only need to use $x + 1$ after that. There is no need for $x + 2$. To show this, assume that the sender has sent the packet with sequence number $x$. Three things can happen.

1. The packet arrives safe and sound at the receiver site; the receiver sends an acknowledgment. The acknowledgment arrives at the sender site, causing the sender to send the next packet numbered $x + 1$.
2. The packet is corrupted or never arrives at the receiver site; the sender resends the packet (numbered $x$) after the time-out. The receiver returns an acknowledgment.
3. The packet arrives safe and sound at the receiver site; the receiver sends an acknowledgment, but the acknowledgment is corrupted or lost. The sender resends the packet (numbered $x$) after the time-out. Note that the packet here is a duplicate. The receiver can recognize this fact because it expects packet $x + 1$ but packet $x$ was received.

We can see that there is a need for sequence numbers $x$ and $x + 1$ because the receiver needs to distinguish between case 1 and case 3. But there is no need for a packet to be numbered $x + 2$. In case 1, the packet can be numbered $x$ again because packets $x$ and $x + 1$ are acknowledged and there is no ambiguity at either site. In cases 2 and 3, the new packet is $x + 1$, not $x + 2$. If only $x$ and $x + 1$ are needed, we can let $x = 0$ and $x + 1 = 1$. This means that the sequence is 0, 1, 0, 1, 0, and so on. This is referred to as modulo 2 arithmetic.

**In the Stop-and-Wait protocol, we can use a 1-bit field to number the packets. The sequence numbers are based on modulo-2 arithmetic.**

### *Acknowledgment Numbers*

Since the sequence numbers must be suitable for both data packets and acknowledgments, we use this convention: The acknowledgment numbers always announce the sequence number of the *next packet expected* by the receiver. For example, if packet 0 has arrived safe and sound, the receiver sends an ACK with acknowledgment 1 (meaning packet 1 is expected next). If packet 1 has arrived safe and sound, the receiver sends an ACK with acknowledgment 0 (meaning packet 0 is expected).

**In the Stop-and-Wait protocol, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next packet expected.**

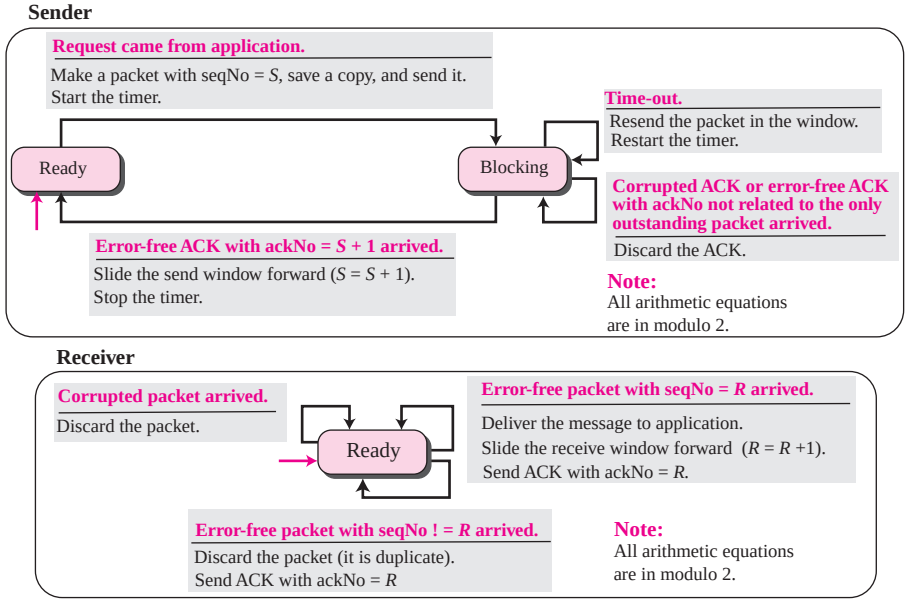
The sender has a control variable, which we call *S* (sender), that points to the only slot in the send window. The receiver has a control variable, which we call *R* (receiver), that points to the only slot in the receive window.

**All calculation in the Stop-and-Wait protocol is in modulo 2.**

### *FSMs*

Figure 13.20 shows the FSMs for the Stop-and-Wait protocol. Since the protocol is a connection-oriented protocol, both ends should be in the *established* state before exchanging data packets. The states we describe here are actually nested in the *established* state.

**Figure 13.20** *FSM for the Stop-and-Wait protocol*




**Sender** The sender is initially in the ready state, but it can move between the ready and blocking state. The variable *S* is initialized to 0.

- **Ready State.** When the sender is in this state, it is only waiting for one event to occur. If a request comes from the application, the sender creates a packet with the sequence number set to *S*. A copy of the packet is stored, and the packet is sent. The sender then starts the only timer. The sender then moves to the blocking state.
- **Blocking State.** When the sender is in this state, three events can occur:

  1. If an error-free ACK arrives with ackNo related to the next packet to be sent, which means ackNo = (*S* + 1) modulo 2, then the timer is stopped. The window is slided, *S* = (*S* + 1) modulo 2. Finally, the sender moves to the ready state.

2. If a corrupted ACK or an error-free ACK with the ackNo ≠ (S + 1) modulo 2 arrives, the ACK is discarded.
3. If a time-out occurs, the sender resends the only outstanding packet and restarts the timer.

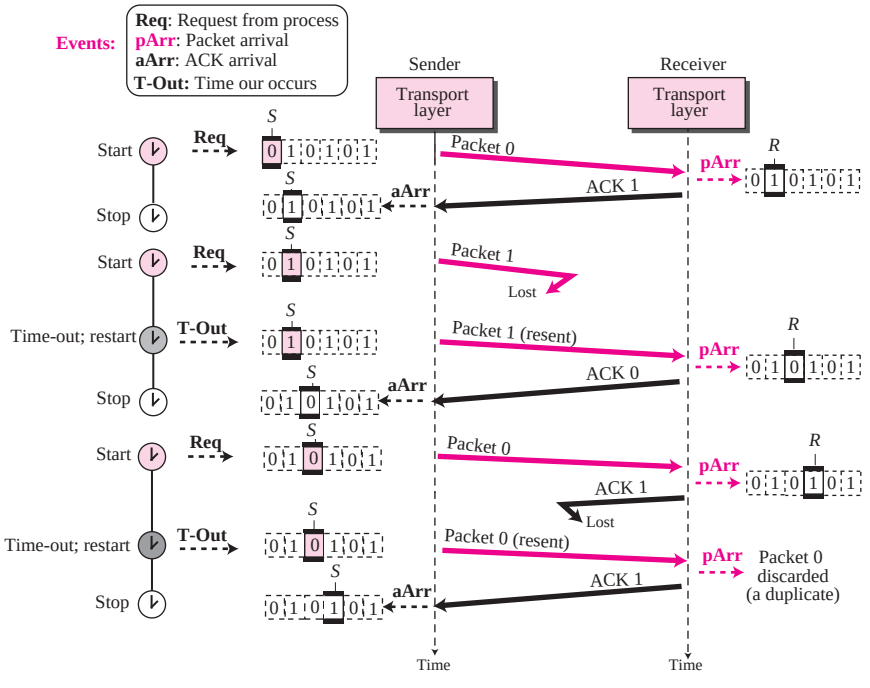
**Receiver** The receiver is always in the *ready* state. The variable R is initialized to 0. Three events may occur:

1. If an error-free packet with seqNo = *R* arrives, the message in the packet is delivered to the application layer. The window then slides, *R* = (*R* + 1) modulo 2. Finally an ACK with ackNo = *R* is sent.
2. If an error-free packet with seqNo ≠ *R* arrives. The packet is discarded, but an ACK with ackNo = *R* is sent.
3. If a corrupted packet arrives, the packet is discarded.

### Example 13.4

Figure 13.21 shows an example of Stop-and-Wait protocol. Packet 0 is sent and acknowledged. Packet 1 is lost and resent after the time-out. The resent packet 1 is acknowledged and the timer stops. Packet 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the packet or the acknowledgment is lost, so after the time-out, it resends packet 0, which is acknowledged.

**Figure 13.21** *Flow diagram for Example 13.4*



### *Efficiency*

The Stop-and-Wait protocol is very inefficient if our channel is *thick* and *long*. By *thick*, we mean that our channel has a large bandwidth (high data rate); by *long*, we mean the round-trip delay is long. The product of these two is called the **bandwidth-delay product.** We can think of the channel as a pipe. The bandwidth-delay product then is the volume of the pipe in bits. The pipe is always there. If we do not use it, we are inefficient. The bandwidth-delay product is a measure of the number of bits a sender can transmit through the system while waiting for an acknowledgment from the receiver.

#### Example 13.5

Assume that, in a Stop-and-Wait system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 milliseconds to make a round trip. What is the bandwidth-delay product? If the system data packets are 1,000 bits in length, what is the utilization percentage of the link?

**Solution**

The bandwidth-delay product is $(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000$ bits. The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and the acknowledgment to come back. However, the system sends only 1,000 bits. We can say that the link utilization is only 1,000/20,000, or 5 percent. For this reason, for a link with a high bandwidth or long delay, the use of Stop-and-Wait wastes the capacity of the link.

#### Example 13.6

What is the utilization percentage of the link in Example 13.5 if we have a protocol that can send up to 15 packets before stopping and worrying about the acknowledgments?

**Solution**

The bandwidth-delay product is still 20,000 bits. The system can send up to 15 packets or 15,000 bits during a round trip. This means the utilization is 15,000/20,000, or 75 percent. Of course, if there are damaged packets, the utilization percentage is much less because packets have to be resent.

### *Pipelining*

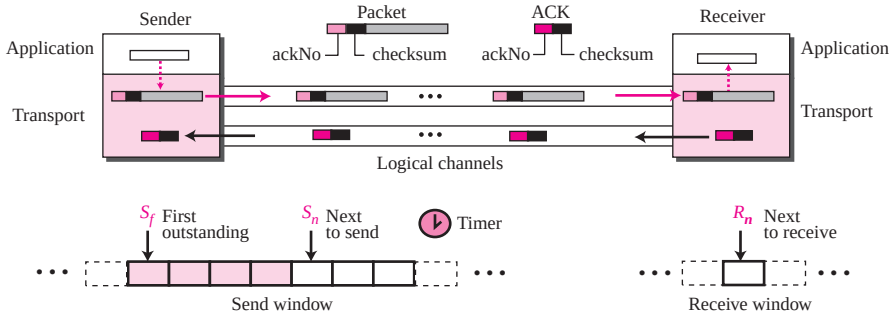
In networking and in other areas, a task is often begun before the previous task has ended. This is known as **pipelining.** There is no pipelining in the Stop-and-Wait protocol because a sender must wait for a packet to reach the destination and be acknowledged before the next packet can be sent. However, pipelining does apply to our next two protocols because several packets can be sent before a sender receives feedback about the previous packets. Pipelining improves the efficiency of the transmission if the number of bits in transition is large with respect to the bandwidth-delay product.

## Go-Back-*N* Protocol

To improve the efficiency of transmission (fill the pipe), multiple packets must be in transition while the sender is waiting for acknowledgment. In other words, we need to let more than one packet be outstanding to keep the channel busy while the sender is waiting for acknowledgment. In this section, we discuss one protocol that can achieve this goal; in the next section, we discuss a second. The first is called **Go-Back-*N* (GBN)** (the

rationale for the name will become clear later). The key to Go-back-*N* is that we can send several packets before receiving acknowledgments, but the receiver can only buffer one packet. We keep a copy of the sent packets until the acknowledgments arrive. Figure 13.22 shows the outline of the protocol. Note that several data packets and acknowledgments can be in the channel at the same time.

**Figure 13.22** *Go-Back-*N *protocol*




### *Sequence Numbers*

As we mentioned before, the sequence numbers are used modulo $2^m$, where $m$ is the size of the sequence number field in bits.

**In the Go-Back-*N* Protocol, the sequence numbers are modulo $2^m$, where *m* is the size of the sequence number field in bits.**

### *Acknowledgment Number*

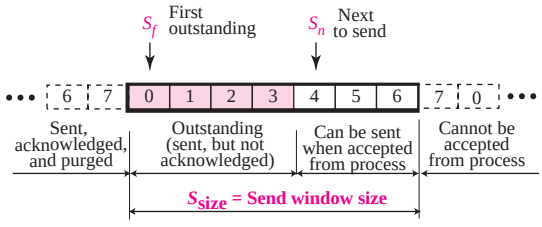
Acknowledgment number in this protocol is cumulative and defines the sequence number of the next packet expected. For example, if the acknowledgment number (ackNo) is 7, it means all packets with sequence number up to 6 have arrived, safe and sound, and the receiver is expecting the packet with sequence number 7.

**In the Go-Back-*N* protocol, the acknowledgment number is cumulative and defines the sequence number of the next packet expected to arrive.**

### *Send Window*

The send window is an imaginary box covering the sequence numbers of the data packets that can be in transit or can be sent. In each window position, some of these sequence numbers define the packets that have been sent; others define those that can be sent. The maximum size of the window is $2^m - 1$ for reasons that we discuss later. In this chapter, we let the size be fixed and set to the maximum value, but we will see in future chapters that some protocols may have a variable window size. Figure 13.23 shows a sliding window of size 7 ($m = 3$) for the Go-Back-*N* protocol.

**Figure 13.23** *Send window for Go-Back-*N




The send window at any time divides the possible sequence numbers into four regions. The first region, left of the window, defines the sequence numbers belonging to packets that are already acknowledged. The sender does not worry about these packets and keeps no copies of them. The second region, colored, defines the range of sequence numbers belonging to the packets that are sent, but have an unknown status. The sender needs to wait to find out if these packets have been received or were lost. We call these *outstanding* packets. The third range, white in the figure, defines the range of sequence numbers for packets that can be sent; however, the corresponding data have not yet been received from the application layer. Finally, the fourth region, right of the window, defines sequence numbers that cannot be used until the window slides.

The window itself is an abstraction; three variables define its size and location at any time. We call these variables $S_f$ (send window, the first outstanding packet), $S_n$ (send window, the next packet to be sent), and $S_{size}$ (send window, size). The variable $S_f$ defines the sequence number of the first (oldest) outstanding packet. The variable $S_n$ holds the sequence number that will be assigned to the next packet to be sent. Finally, the variable $S_{size}$ defines the size of the window, which is fixed in our protocol.

**The send window is an abstract concept defining an imaginary box of maximum size = $2^m - 1$ with three variables: $S_f$, $S_n$, and $S_{size}$.**

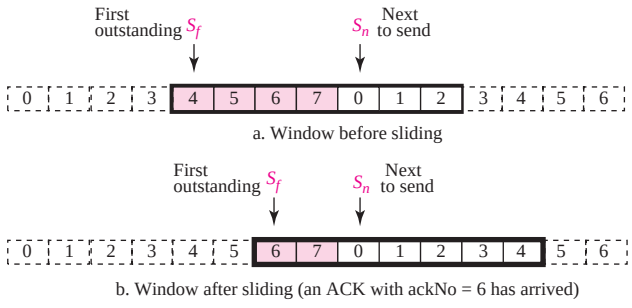
Figure 13.24 shows how a send window can slide one or more slots to the right when an acknowledgment arrives from the other end. In the figure, an acknowledgment with ackNo = 6 has arrived. This means that the receiver is waiting for packets with sequence number 6.

**The send window can slide one or more slots when an error-free ACK with ackNo between $S_f$ and $S_n$ (in modular arithmetic) arrives.**

### *Receive Window*

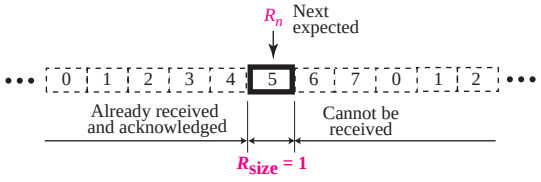
The receive window makes sure that the correct data packets are received and that the correct acknowledgments are sent. In Go-back-*N*, the size of the receive window is always 1. The receiver is always looking for the arrival of a specific packet. Any packet arriving out of order is discarded and needs to be resent. Figure 13.25 shows the

**Figure 13.24** *Sliding the send window*




receive window. Note that we need only one variable $R_n$ (receive window, next packet expected) to define this abstraction. The sequence numbers to the left of the window belong to the packets already received and acknowledged; the sequence numbers to the right of this window define the packets that cannot be received. Any received packet with a sequence number in these two regions is discarded. Only a packet with a sequence number matching the value of $R_n$ is accepted and acknowledged. The receive window also slides, but only one slot at a time. When a correct packet is received, the window slides, $R_n = (R_n + 1)$ modulo $2^m$.

**Figure 13.25** *Receive window for Go-Back-*N




**The receive window is an abstract concept defining an imaginary box of size 1 with one single variable $R_n$. The window slides when a correct packet has arrived; sliding occurs one slot at a time.**

### *Timers*

Although there can be a timer for each packet that is sent, in our protocol we use only one. The reason is that the timer for the first outstanding packet always expires first. We resend all outstanding packets when this timer expires.

### *Resending packets*

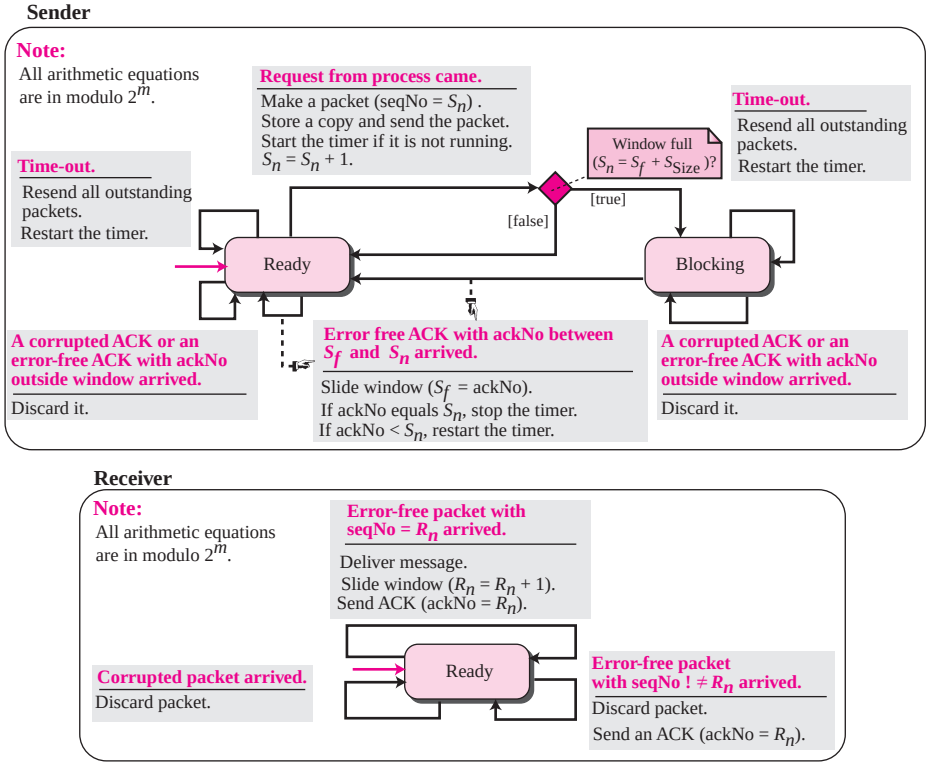
When the timer expires, the sender resends all outstanding packets. For example, suppose the sender has already sent packet 6 ($S_n = 7$), but the only timer expires. If $S_f = 3$,

this means that packets 3, 4, 5, and 6 have not been acknowledged; the sender goes back and resends packets 3, 4, 5, and 6. That is why the protocol is called Go-Back-*N*. On a time-out, the machine goes back *N* locations and resends all packets.

### *FSMs*

Figure 13.26 shows the FSMs for the GBN protocol.

**Figure 13.26** *FSM for the Go-Back-*N *protocol*




**Sender** The sender starts in the ready state, but it can be in one of the two states after: *ready* and *blocking*. The two variables are normally initialized to 0 ($S_f = S_n = 0$), but we will see in the future chapters that some protocols in the TCP/IP protocol use a different initialization.

- **Ready State**. Four events may occur when the sender is in ready state.

  1. If a request comes from the application layer, the sender creates a packet with the sequence number set to $S_n$. A copy of the packet is stored, and the packet is sent. The sender also starts the only timer if it is not running. The value of $S_n$ is now incremented, ($S_n = S_n + 1$) modulo $2^m$. If the window is full, $S_n = (S_f + S_{size})$ modulo $2^m$, the sender goes to the blocking state.

2. If an error-free ACK arrives with ackNo related to one of the outstanding packets, the sender slides the window (set $S_f$ = ackNo) and if all outstanding packets are acknowledged (ackNo = $S_n$), then the timer is stopped. If all outstanding packets are not acknowledged, the timer is restarted.
3. If a corrupted ACK or an error-free ACK with ackNo not related to the outstanding packet arrives, it is discarded.
4. If a time-out occurs, the sender resends all outstanding packet and restarts the timer.

- **Blocking State.** Three events may occur in this case:
  1. If an error-free ACK arrives with ackNo related to one of the outstanding packets, the sender slides the window (set $S_f$ = ackNo) and if all outstanding packets are acknowledged (ackNo = $S_n$), then the timer is stopped. If all outstanding packets are not acknowledged, the timer is restarted. The sender then moves to the ready state.
  2. If a corrupted ACK or an error-free ACK with the ackNo not related to outstanding packets arrives, the ACK is discarded.
  3. If a time-out occurs, the sender sends all outstanding packets and restarts the timer.

**Receiver** The receiver is always in the *ready* state. The only variable $R_n$ is initialized to 0. Three events may occur:

1. If an error-free packet with seqNo = $R_n$ arrives, the message in the packet is delivered to the application layer. The window then slides, $R_n = (R_n + 1)$ modulo $2^m$. Finally an ACK is sent with acqNo = $R_n$.
2. If an error-free packet with seqNo outside the window arrives, the packet is discarded, but an ACK with ackNo = $R_n$ is sent.
3. If a corrupted packet arrives, it is discarded.

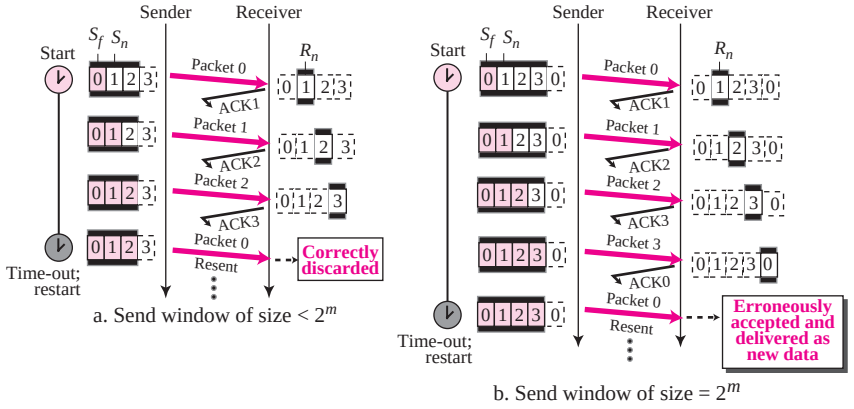
### *Send Window Size*

We can now show why the size of the send window must be less than $2^m$. As an example, we choose $m = 2$, which means the size of the window can be $2^m - 1$, or 3. Figure 13.27 compares a window size of 3 against a window size of 4.

If the size of the window is 3 (less than $2^m$) and all three acknowledgments are lost, the only timer expires and all three packets are resent. The receiver is now expecting packet 3, not packet 0, so the duplicate packet is correctly discarded. On the other hand, if the size of the window is 4 (equal to $2^2$) and all acknowledgments are lost, the sender will send a duplicate of packet 0. However, this time the window of the receiver expects to receive packet 0 (in the next cycle), so it accepts packet 0, not as a duplicate, but as the first packet in the next cycle. This is an error. This shows that the size of the send window must be less than $2^m$.

**In the Go-Back-*N* protocol, the size of the send window must be less than $2^m$; the size of the receive window is always 1.**

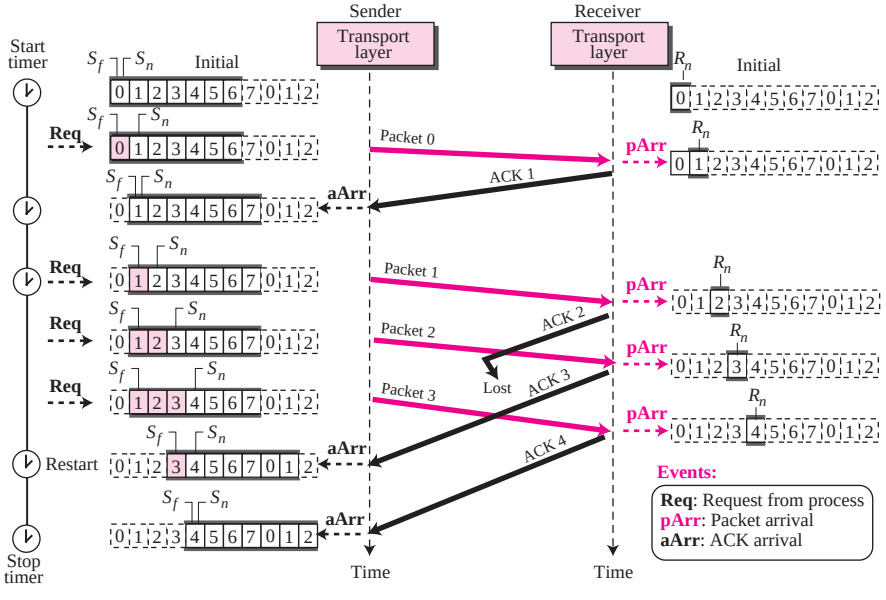
**Figure 13.27** *Send window size for Go-Back-N*




## Example 13.7

Figure 13.28 shows an example of Go-Back-*N*. This is an example of a case where the forward channel is reliable, but the reverse is not. No data packets are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost.

**Figure 13.28** *Flow diagram for Example 13.7*

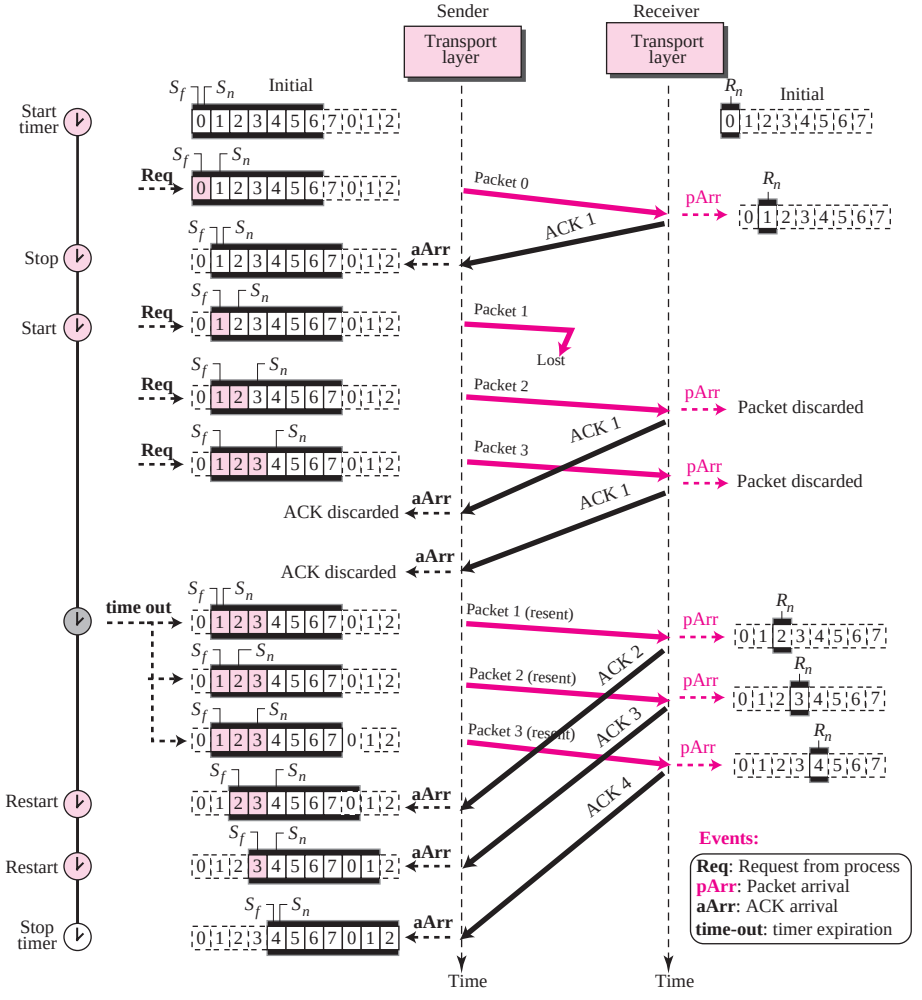


After initialization, there are some sender events. Request events are triggered by message chunks from the application layer; arrival events are triggered by ACK received from the network layer. There is no time-out event here because all outstanding packets are acknowledged before the timer expires. Note that although ACK 2 is lost, ACK 3 is cumulative and serves as both ACK 2 and ACK 3. There are four events at the receiver site.

### Example 3.8

Figure 13.29 shows what happens when a packet is lost. Packets 0, 1, 2, and 3 are sent. However, packet 1 is lost. The receiver receives packets 2 and 3, but they are discarded because they are received out of order (packet 1 is expected). When the receiver receives packets 2 and 3, it sends ACK1 to show that it expects to receive packet 1. However, these ACKs are not useful for the sender because the ackNo is equal $S_f$, not greater that $S_f$. So the sender discards them. When the time-out occurs, the sender resends packets 1, 2, and 3, which are acknowledged.

**Figure 13.29** *Flow diagram for Example 3.8*



### *Go-Back-N versus Stop-and-Wait*

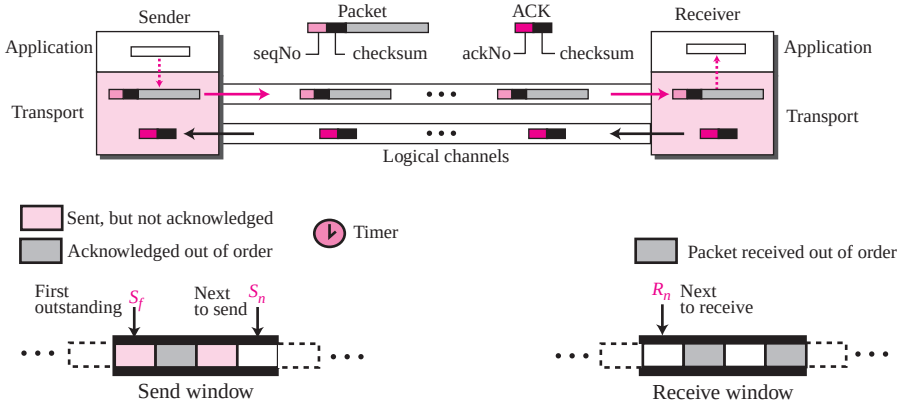
The reader may find that there is a similarity between the Go-Back-*N* protocol and the Stop-and-Wait protocol.The Stop-and-Wait protocol is actually a Go-Back-*N* protocol in which there are only two sequence numbers and the send window size is 1. In other words, $m = 1$ and $2^m - 1 = 1$. In Go-Back-*N*, we said that the arithmetic is modulo $2^m$; in Stop-and-Wait it is modulo 2, which is the same as $2^m$ when $m = 1$.

## Selective-Repeat Protocol

The Go-Back-*N* protocol simplifies the process at the receiver. The receiver keeps track of only one variable, and there is no need to buffer out-of-order packets; they are simply discarded. However, this protocol is inefficient if the underlying network protocol loses a lot of packets. Each time a single packet is lost or corrupted, the sender resends all outstanding packets although some of these packets may have been received safe and sound, but out of order. If the network layer is losing many packets because of congestion in the network, the resending of all of these outstanding packets makes the congestion worse, and eventually more packets are lost. This has an avalanche effect that may result in the total collapse of the network.

Another protocol, called the **Selective-Repeat (SR) protocol,** has been devised that, as the name implies, resends only selective packets, those that are actually lost. The outline of this protocol is shown in Figure 13.30.

**Figure 13.30** *Outline of Selective-Repeat*

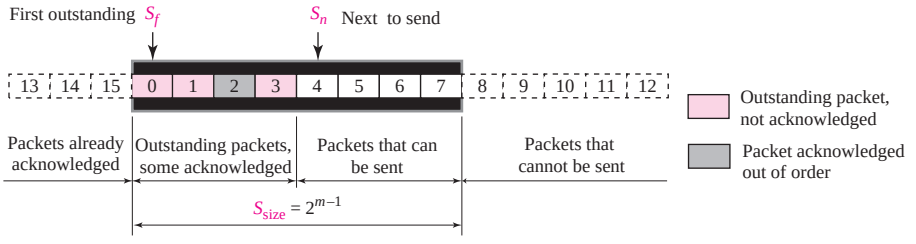



### *Windows*

The Selective-Repeat protocol also uses two windows: a send window and a receive window. However, there are differences between the windows in this protocol and the ones in Go-Back-*N*. First, the maximum size of the send window is much smaller; it is $2^{m-1}$. The reason for this will be discussed later. Second, the receive window is the same size as the send window.

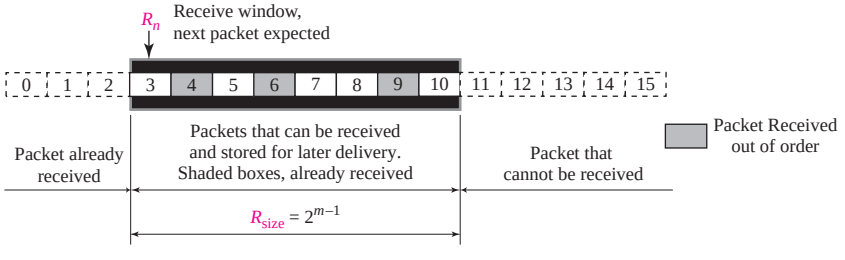
The send window maximum size can be $2^{m-1}$. For example, if $m = 4$, the sequence numbers go from 0 to 15, but the maximum size of the window is just 8 (it is 15 in the Go-Back-*N* Protocol). We show the Selective-Repeat send window in Figure 13.31 to emphasize the size.

**Figure 13.31** *Send window for Selective-Repeat protocol*




The receive window in Selective-Repeat is totally different from the one in Go-Back-*N*. The size of the receive window is the same as the size of the send window (maximum $2^{m-1}$). The Selective-Repeat protocol allows as many packets as the size of the receive window to arrive out of order and be kept until there is a set of consecutive packets to be delivered to the application layer. Because the sizes of the send window and receive window are the same, all the packets in the send packet can arrive out of order and be stored until they can be delivered. We need, however, to emphasize that in a reliable protocol, the receiver *never* delivers packets out of order to the application layer. Figure 13.32 shows the receive window in the Selective-Repeat. Those slots inside the window that are shaded define packets that have arrived out of order and are waiting for the earlier transmitted packet to arrive before delivery to the application layer.

**Figure 13.32** *Receive window for Selective-Repeat protocol*




### *Timer*

Theoretically, Selective-Repeat uses one timer for each outstanding packet. When a timer expires, only the corresponding packet is resent. In other words, GBN treats outstanding packets as a group; SR treats them individually. However, most transport layer protocol that implement SR use only one single timer. For this reason, we use only one timer.

### *Acknowledgments*

Still there is another difference between the two protocols. In GBN an ackNo is cumulative; it defines the sequence number of the next packet expected, confirming that all previous packets have been received safe and sound. The semantics of acknowledgment is different in SR. In SR, an ackNo defines the sequence number of one single packet that is received safe and sound; there is no feedback for any other.

**In the Selective-Repeat protocol, an acknowledgment number defines the sequence number of the error-free packet received.**

### Example 13.9

Assume a sender sends 6 packets: packets 0, 1, 2, 3, 4, and 5. The sender receives an ACK with ackNo = 3. What is the interpretation if the system is using GBN or SR?

**Solution**

If the system is using GBN, it means that packets 0, 1, and 2 have been received uncorrupted and the receiver is expecting packet 3. If the system is using SR, it means that packet 3 has been received uncorrupted; the ACK does not say anything about other packets.

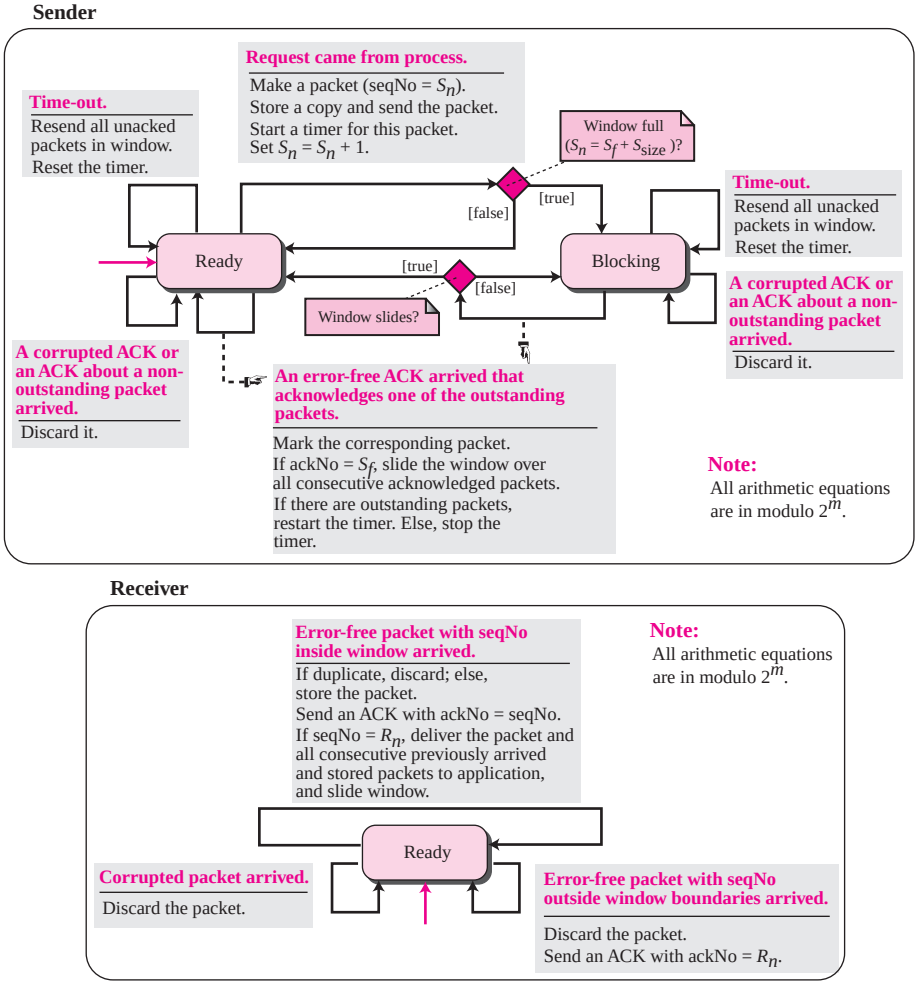
### *FSMs*

Figure 13.33 shows the FSMs for the Selective-Repeat protocol. It is similar to the ones for the GBN, but there are some differences.

**Sender** The sender starts in the *ready* state, but later it can be in one of the two states: *ready* or *blocking*. The following shows the events and the corresponding actions in each state.

- **Ready State.** Four events may occur in this case:
  1. If a request comes from the application layer, the sender creates a packet with the sequence number set to $S_n$. A copy of the packet is stored, and the packet is sent. If the timer is not running, the sender starts the timer. The value of $S_n$ is now incremented, $S_n = (S_n + 1)$ modulo $2^m$. If the window is full, $S_n = (S_f + S_{size})$ modulo $2^m$, the sender goes to the blocking state.
  2. If an error-free ACK arrives with ackNo related to one of the outstanding packets, that packet is marked as acknowledged. If the ackNo = $S_f$, the window slides to the right until the $S_f$ points to the first unacknowledged packet (all consecutive acknowledged packets are now outside the window). If there are outstanding packets, restarts the timer; else, stops the timer.
  3. If a corrupted ACK or an error-free ACK with ackNo not related to an outstanding packet arrives, it is discarded.
  4. If a time-out occurs, the sender resends all unacknowledged packets in the window and restarts the timer.
- **Blocking State.** Three events may occur in this case:
  1. If an error-free ACK arrives with ackNo related to one of the outstanding packets, that packet is marked as acknowledged. In addition, if the ackNo = $S_f$, the

**Figure 13.33** *FSMs for SR protocol*




window is slid to the right until the $S_f$ points to the first unacknowledged packet (all consecutive acknowledged packets are now outside the window). If the window has slid, the sender moves to the ready state.

2. If a corrupted ACK or an error-free ACK with the ackNo not related to outstanding packets arrives, the ACK is discarded.
3. If a time-out occurs, the sender resends all unacknowledged packets in the window and restarts the timer.

**Receiver** The receiver is always in the *ready* state. Three events may occur:

1. If an error-free packet with seqNo in the window arrives, the packet is stored and an ACK with ackNo = seqNo is sent. In addition, if the seqNo = $R_n$, then the packet

and all previously arrived consecutive packets are delivered to the application layer and the window slides so that the $R_n$ points to the first empty slot.

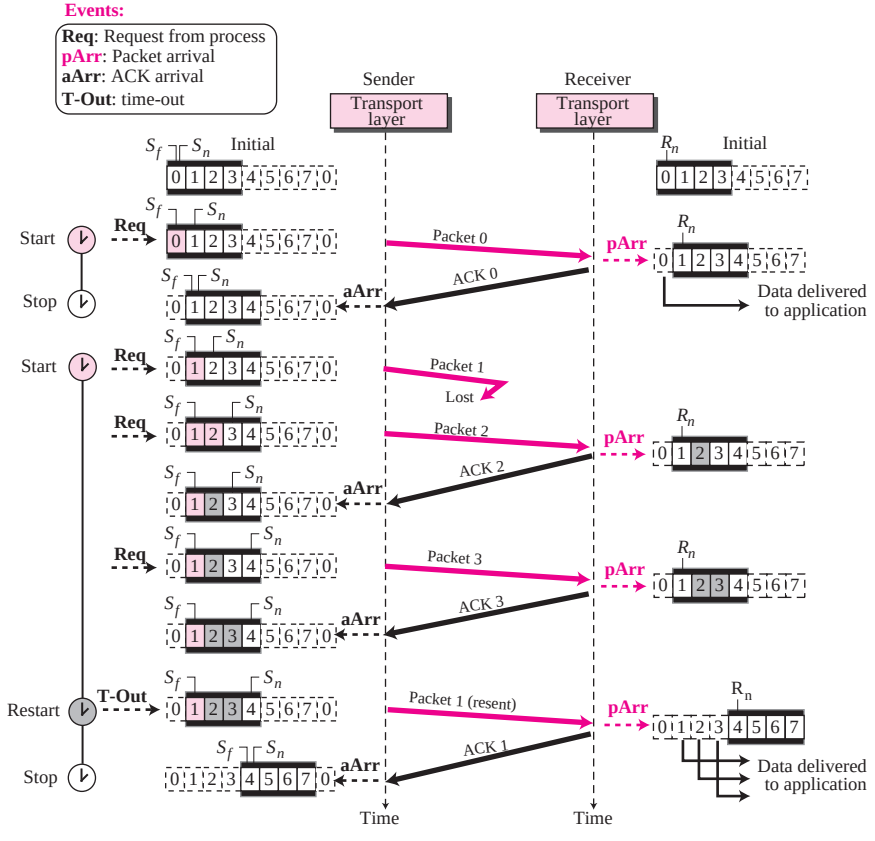
2. If an error-free packet with seqNo outside the window arrives, the packet is discarded, but an ACK with ackNo = $R_n$ is returned to the sender. This is needed to let the sender slide its window if some ACKs related to packets with seqNo < $R_n$ were lost.
3. If a corrupted packet arrives, the packet is discarded.

### Example 13.10

This example is similar to Example 3.8 (Figure 13.29) in which packet 1 is lost. We show how Selective-Repeat behaves in this case. Figure 13.34 shows the situation.

At the sender, packet 0 is transmitted and acknowledged. Packet 1 is lost. Packets 2 and 3 arrive out of order and are acknowledged. When the timer times out, packet 1 (the only unacknowledged packet) is resent and is acknowledged. The send window then slides.

**Figure 13.34** *Flow diagram for Example 13.10*

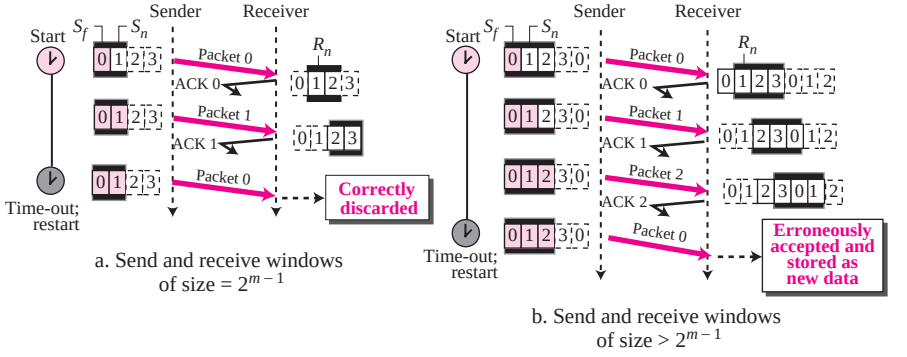


At the receiver site we need to distinguish between the acceptance of a packet and its delivery to the application layer. At the second arrival, packet 2 arrives and is stored and marked (shaded slot), but it cannot be delivered because packet 1 is missing. At the next arrival, packet 3 arrives and is marked and stored, but still none of the packets can be delivered. Only at the last arrival, when finally a copy of packet 1 arrives, can packets 1, 2, and 3 be delivered to the application layer. There are two conditions for the delivery of packets to the application layer: First, a set of consecutive packets must have arrived. Second, the set starts from the beginning of the window. After the first arrival, there was only one packet and it started from the beginning of the window. After the last arrival, there are three packets and the first one starts from the beginning of the window. The key is that a reliable transport layer promises to deliver packets *in order*.

### *Window Sizes*

We can now show why the size of the sender and receiver windows can be at most one-half of $2^m$. For an example, we choose $m = 2$, which means the size of the window is $2^m/2$, or 2. Figure 13.35 compares a window size of 2 with a window size of 3.

**Figure 13.35** *Selective-Repeat, window size*




If the size of the window is 2 and all acknowledgments are lost, the timer for packet 0 expires and packet 0 is resent. However, the window of the receiver is now expecting packet 2, not packet 0, so this duplicate packet is correctly discarded (the sequence number 0 is not in the window). When the size of the window is 3 and all acknowledgments are lost, the sender sends a duplicate of packet 0. However, this time, the window of the receiver expects to receive packet 0 (0 is part of the window), so it accepts packet 0, not as a duplicate, but as a packet in the next cycle. This is clearly an error.

**In Selective-Repeat, the size of the sender and receiver window can be at most one-half of $2^m$.**

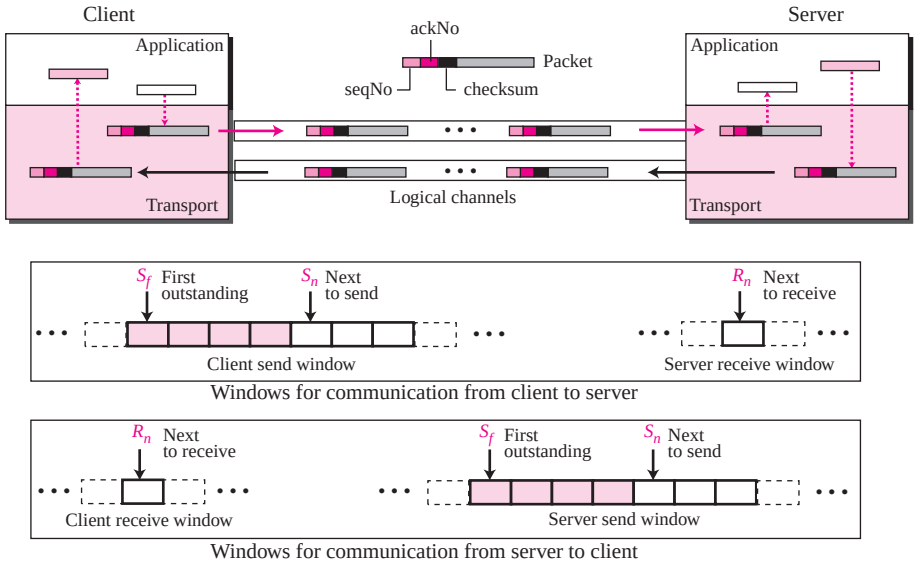
## Bidirectional Protocols: Piggybacking

The four protocols we discussed in this section are all unidirectional: data packets flow in only one direction and acknowledgments travel in the other direction. In real life, data packets are normally flowing in both directions: from client to server and from server to client. This means that acknowledgments also need to flow in both directions.

A technique called **piggybacking** is used to improve the efficiency of the bidirectional protocols. When a packet is carrying data from A to B, it can also carry acknowledgment feedback about arrived packets from B; when a packet is carrying data from B to A, it can also carry acknowledgment feedback about the arrived packets from A.

Figure 13.36 shows the layout for the GBN protocol implemented bidirectionally using piggybacking. The client and server each use two independent windows: send and receive windows. We leave the FSMs for this case, and others, as exercises.

**Figure 13.36** *Design of piggybacking in Go-Back-N*




## 13.3 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books. The items enclosed in brackets refer to the reference list at the end of the book: [Com 06], [Pet & Dav 03], [Kur & Ros 08], [Gar & Vid 04], [Far 04], [Tan 03], and [Sta 04]. In addition, we recommend the following informative paper: "The Transport Layer: Tutorial and Survey" by Sami Iren, Paul D. Amer, and Phillip T. Conrad, *ACM Computing Surveys*, vol. 31, no. 4, Dec. 1999.

## 13.4 KEY TERMS

acknowledgment number
bandwidth-delay product
client-server paradigm
closed-loop congestion control
congestion
congestion control
demultiplexing
ephemeral port number

finite state machine
Go-back-*N* protocol
load
multiplexing
open-loop congestion control
piggybacking
pipelining
port number
process-to-process communication
Selective-Repeat protocol
sequence number
sliding window
socket address
Stop-and-Wait protocol
well-known port number

## 13.5 SUMMARY

- The main duty of a transport-layer protocol is to provide process-to-process communication. To define the processes, we need port numbers. The client program defines itself with an ephemeral port number. The server defines itself with a well-known port number.
- To send a message from one process to another, the transport layer protocol encapsulates and decapsulates messages. Encapsulation happens at the sender site. Decapsulation happens at the receiver site.
- The transport layer at the source performs multiplexing, collecting messages from server processes for transmission; the transport layer at the destination performs demultiplexing, delivering messages to different processes.
- Flow control balances the exchange of data items between a producer and a consumer. At the transport layer, we use buffers to hold packets when the consumer is not ready to accept them. Reliability at the transport layer can be achieved by adding error control, which includes detection of corrupted packets, resending lost and corrupted packets, discarding duplicate packets, and reordering packets that arrived out of order. To manage flow and error control, we use sequence numbers to number packets and use acknowledgment numbers to refer to the numbered packets.
- A transport layer can provide two types of congestion control: open-loop and closed loop. In an open-loop congestion control, the protocol tries to avoid the congestion; in closed-loop congestion control, the protocol tries to detect and remove the congestion after it has occurred.
- A transport-layer protocol can provide two types of services: connectionless and connection-oriented. In a connectionless service, the sender sends packets to the receiver without any connection establishment. In a connection-oriented service, the client and the server first need to establish a connection between themselves.The data exchange can only happen after the connection establishment. After data exchange, the connection needs to be torn down.
- We have discussed several common transport-layer protocols in this chapter. The simple connectionless protocol provides neither flow control nor error control. The connection-oriented Stop-and-Wait protocol provides both flow and error control, but is inefficient. The Go-back-*N* protocol is the more efficient version of the Stop-and-Wait protocol that takes advantage of pipelining. The Selective-Repeat protocol

is a modification of the Go-back-*N* protocol that is better suited to handle packet loss. All of these protocols can be implemented bidirectionally using piggybacking.

## 13.6 PRACTICE SET

### Exercises

1. A sender sends a series of packets to the same destination using 5-bit sequence of numbers. If the sequence number starts with 0, what is the sequence number of the 100th packet?
2. Using 5-bit sequence numbers, what is the maximum size of the send and receive windows for each of the following protocols?
   a. Stop-and-Wait
   b. Go-Back-*N*
   c. Selective-Repeat
3. Show the FSM for an imaginary machine with three states: state A (starting state), state B, and state C; and four events: events 1, 2, 3, and 4. The following specify the behavior of the machine:
   a. When in state A, two events may occur: event 1 and event 2. If event 1 occurs, the machine performs action 1 and moves to state B. If event 2 occurs, the machine moves to state C (no action).
   b. When in state B, two events may occur: event 3 and event 4. If event 3 occurs, the machine performs action 2, but remains in state B. If event 4 occurs, the machine just moves to state C.
   c. When in state C, the machine remains in this state forever.
4. Redesign the FSM in Figure 13.15 if the connection establishment is done with only three packet exchange (combining packet 2 and 3).
5. Redraw Figure 13.18 with 5 packets exchanged (0, 1, 2, 3, 4). Assume packet 2 is lost and packet 3 arrives after packet 4.
6. Create a scenario similar to Figure 13.21 in which the sender sends three packets. The first and second packets arrived and acknowledged. The third packet is delayed and resent. The duplicate packet is received after the acknowledgment for the original to be sent.
7. Create a scenario similar to figure 13.21 in which the sender sends two packets. The first packet is received and acknowledged, but the acknowledgement is lost. The sender resends the packet after time-out. The second packet is lost and resent.
8. Redraw Figure 13.28 if the sender sends 5 packets (0, 1, 2, 3, and 4). Packets 0, 1, and 2 are sent and acknowledged in one single ACK, which arrives at the sender site after all packets have been sent. Packet 3 is received and acknowledged in a single ACK. Packet 4 is lost and resent.
9. Redraw Figure 13.34 if the sender sends 5 packets (0, 1, 2, 3, and 4). Packets 0, 1, and 2 are received in order and acknowledged, one by one. Packet 3 is delayed and received after packet 4.

**10.** Answer the following questions related to the FSMs for the Stop-and-Wait protocol (Figure 13.20):

- **a.** The sending machine is in the ready state and $S = 0$. What is the sequence number of the next packet to send?
- **b.** The sending machine is in the blocking state and $S = 1$. What is the sequence number of the next packet to send if a time-out occurs.
- **c.** The receiving machine is in the ready state and $R = 1$. A packet with the sequence number 1 arrives. What is the action in response to this event?
- **d.** The receiving machine is in the ready state and $R = 1$. A packet with the sequence number 0 arrives. What is the action in response to this event?

**11.** Answer the following questions related to the FSM's for the Go-back-$N$ protocol with $m = 6$ (Figure 13.26):

- **a.** The sending machine is in the ready state with $S_f = 10$ and $S_n = 15$. What is the sequence number of the next packet to send?
- **b.** The sending machine is in the ready state with $S_f = 10$ and $S_n = 15$. A time-out occurs. How many packets are to be resent? What are their sequence numbers?
- **c.** The sending machine is in the ready state with $S_f = 10$ and $S_n = 15$. An ACK with ackNo = 13 arrives. What are the next values of $S_f$ and $S_n$?
- **d.** The sending machine is in the blocking state with $S_f = 14$ and $S_n = 21$. What is the size of the window?
- **e.** The sending machine is in the blocking state with $S_f = 14$ and $S_n = 21$. An ACK with ackNo = 18 arrives. What are the next values of $S_f$ and $S_n$? What is the state of the sending machine?
- **f.** The receiving machine is in the ready state with $R_n = 16$. A packet with sequence number 16 arrives. What is the next value of $R_n$? What is the response of the machine to this event?

**12.** Answer the following questions related to the FSM's for the Selective-Repeat protocol with $m = 7$ bits (Figure 13.33):

- **a.** The sending machine is in the ready state with $S_f = 10$ and $S_n = 15$. What is the sequence number of the next packet to send?
- **b.** The sending machine is in the ready state with $S_f = 10$ and $S_n = 15$. The timer for packet 10 times out. How many packets are to be resent? What are their sequence numbers?
- **c.** The sending machine is in the ready state with $S_f = 10$ and $S_n = 15$. An ACK with ackNo = 13 arrives. What are the next values of $S_f$ and $S_n$? What is the action in response to this event?
- **d.** The sending machine is in the blocking state with $S_f = 14$ and $S_n = 21$. What is the size of the window?
- **e.** The sending machine is in the blocking state with $S_f = 14$ and $S_n = 21$. An ACK with ackNo = 14 arrives. Packets 15 and 16 have been already acknowledged. What are the next values of $S_f$ and $S_n$? What is the state of the sending machine?
- **f.** The receiving machine is in the ready state with $R_n = 16$. The size of the window is 8. A packet with sequence number 16 arrives. What is the next value of $R_n$? What is the response of the machine to this event?

**g.** The receiving machine is in the ready state with $R_n = 16$. The size of the window is 8. A packet with sequence number 18 arrives. What is the next value of $R_n$? What is the response of the machine to this event?

**h.** The receiving machine is in the ready state with $R_n = 16$. The size of the window is 8. A packet with sequence number 18 arrives. What is the next value of $R_n$? What is the response of the machine to this event?

## Research Activities

**13.** Redraw the bidirectional outline (using piggybacking) for the simple protocol in Figure 13.16.

**14.** Redraw the bidirectional outline (using piggybacking) for the Stop-and-Wait protocol in Figure 13.19.

**15.** Redraw the bidirectional outline (using piggybacking) for the Selective-Repeat protocol in Figure 13.30.

**16.** Show the bidirectional FSMs for the simple protocol using piggybacking. Note that both parties need to send and receive.

**17.** Show the bidirectional FSMs for the Stop-and-Wait protocol using piggybacking. Note that both parties need to send and receive.

**18.** Show the bidirectional FSMs for the Go-back-*N* protocol using piggybacking. Note that both parties need to send and receive.

**19.** Show the bidirectional FSMs for the Selective-Repeat protocol using piggybacking. Note that both parties need to send and receive.

**20.** Write two algorithms in pseudocode (or in a computer language) for the FSMs related to the simple protocol (Figure 13.17).

**21.** Write two algorithms in pseudocode (or in a computer language) for the FSMs related to the Stop-and-Wait protocol (Figure 13.20).

**22.** Write two algorithms in pseudocode (or in a computer language) for the FSMs related to the Go-back-*N* protocol (Figure 13.26).

**23.** Write two algorithms in pseudocode (or in a computer language) for the FSMs related to the Selective-Repeat protocol (Figure 13.33).

CHAPTER

# 14

# *User Datagram Protocol (UDP)*

The original TCP/IP protocol suite specifies two protocols for the transport layer: UDP and TCP. We first focus on UDP, the simpler of the two, before discussing TCP in Chapter 15. A new transport-layer protocol, SCTP, has been designed, which we will discuss in Chapter 16.

## OBJECTIVE

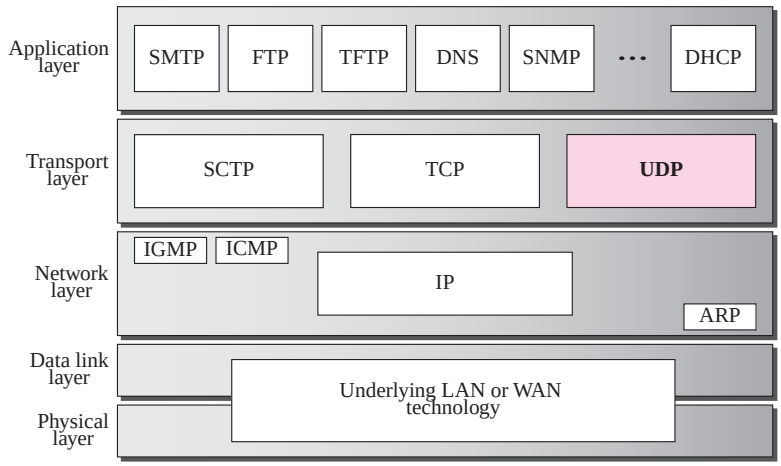
*We have several objectives for this chapter:*

- To introduce UDP and show its relationship to other protocols in the TCP/IP protocol suite.
- To explain the format of a UDP packet, which is called a user datagram, and discuss the use of each field in the header.
- To discuss the services provided by the UDP such as process-to-process delivery, rudimentary error control, multiplexing/demultiplexing, and queuing.
- To show how to calculate the optional checksum and why the sender of a UDP packet needs to add a pseudoheader to the packet when calculating the checksum.
- To discuss how some application programs can benefit from the simplicity of UDP.
- To briefly discuss the structure of a software package that implements UDP and give the description of control-block, input, and output module.

## 14.1 INTRODUCTION

Figure 14.1 shows the relationship of the **User Datagram Protocol** (UDP) to the other protocols and layers of the TCP/IP protocol suite: UDP is located between the application layer and the IP layer, and serves as the intermediary between the application programs and the network operations.

**Figure 14.1** *Position of UDP in the TCP/IP protocol suite*




As discussed in Chapter 13, a transport layer protocol usually has several responsibilities. One is to create a process-to-process communication; UDP uses port numbers to accomplish this. Another responsibility is to provide control mechanisms at the transport level. UDP does this task at a very minimal level. There is no flow control mechanism and there is no acknowledgment for received packets. UDP, however, does provide error control to some extent. If UDP detects an error in the received packet, it silently drops it.

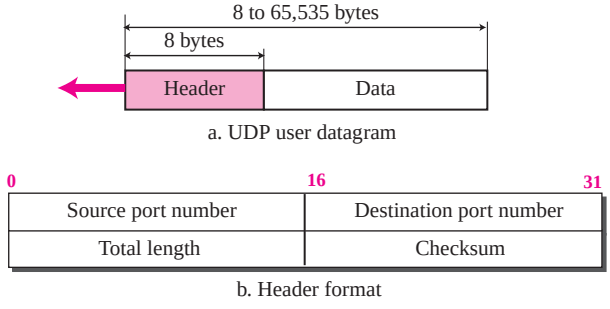
UDP is a **connectionless, unreliable transport protocol.** It does not add anything to the services of IP except for providing process-to-process communication instead of host-to-host communication.

If UDP is so powerless, why would a process want to use it? With the disadvantages come some advantages. UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message using UDP takes much less interaction between the sender and receiver than using TCP.

## 14.2 USER DATAGRAM

UDP packets, called **user datagrams,** have a fixed-size header of 8 bytes. Figure 14.2 shows the format of a user datagram. The fields are as follows:

**Figure 14.2** *User datagram format*




- **Source port number.** This is the port number used by the process running on the source host. It is 16 bits long, which means that the port number can range from 0 to 65,535. If the source host is the client (a client sending a request), the port number, in most cases, is an ephemeral port number requested by the process and chosen by the UDP software running on the source host. If the source host is the server (a server sending a response), the port number, in most cases, is a well-known port number.
- **Destination port number.** This is the port number used by the process running on the destination host. It is also 16 bits long. If the destination host is the server (a client sending a request), the port number, in most cases, is a well-known port number. If the destination host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number. In this case, the server copies the ephemeral port number it has received in the request packet.
- **Length.** This is a 16-bit field that defines the total length of the user datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes. However, the total length needs to be much less because a UDP user datagram is stored in an IP datagram with the total length of 65,535 bytes. The length field in a UDP user datagram is actually not necessary. A user datagram is encapsulated in an IP datagram. There is a field in the IP datagram that defines the total length. There is another field in the IP datagram that defines the length of the header. So if we subtract the value of the second field from the first, we can deduce the length of the UDP datagram that is encapsulated in an IP datagram.

**UDP length = IP length – IP header's length**

However, the designers of the UDP protocol felt that it was more efficient for the destination UDP to calculate the length of the data from the information provided

in the UDP user datagram rather than ask the IP software to supply this information. We should remember that when the IP software delivers the UDP user datagram to the UDP layer, it has already dropped the IP header.

- **Checksum.** This field is used to detect errors over the entire user datagram (header plus data). The checksum is discussed in the next section.

### Example 14.1

The following is a dump of a UDP header in hexadecimal format.

**CB84000D001C001C**

**a.** What is the source port number?
**b.** What is the destination port number?
**c.** What is the total length of the user datagram?
**d.** What is the length of the data?
**e.** Is the packet directed from a client to a server or vice versa?
**f.** What is the client process?

**Solution**

**a.** The source port number is the first four hexadecimal digits ($CB84_{16}$), which means that the source port number is 52100.
**b.** The destination port number is the second four hexadecimal digits ($000D_{16}$), which means that the destination port number is 13.
**c.** The third four hexadecimal digits ($001C_{16}$) define the length of the whole UDP packet as 28 bytes.
**d.** The length of the data is the length of the whole packet minus the length of the header, or 28 – 8 = 20 bytes.
**e.** Since the destination port number is 13 (well-known port), the packet is from the client to the server.
**f.** The client process is the Daytime (see Table 14.1).

## 14.3 UDP SERVICES

We discussed the general services provided by a transport layer protocol in Chapter 13. In this section, we discuss what portions of those general services are provided by UDP.

### Process-to-Process Communication

UDP provides process-to-process communication discussed in Chapter 13 using sockets, a combination of IP addresses and port numbers. Several port numbers used by UDP are shown in Table 14.1.

**Table 14.1** *Well-known Ports used with UDP*

| *Port* | *Protocol* | *Description* |
|---|---|---|
| 7 | Echo | Echoes a received datagram back to the sender |
| 9 | Discard | Discards any datagram that is received |
| 11 | Users | Active users |
| 13 | Daytime | Returns the date and the time |
| 17 | Quote | Returns a quote of the day |
| 19 | Chargen | Returns a string of characters |
| 53 | Domain | Domain Name Service (DNS) |
| 67 | Bootps | Server port to download bootstrap information |
| 68 | Bootpc | Client port to download bootstrap information |
| 69 | TFTP | Trivial File Transfer Protocol |
| 111 | RPC | Remote Procedure Call |
| 123 | NTP | Network Time Protocol |
| 161 | SNMP | Simple Network Management Protocol |
| 162 | SNMP | Simple Network Management Protocol (trap) |

## Connectionless Services

As mentioned previously, UDP provides a *connectionless service.* This means that each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program. The user datagrams are not numbered. Also, there is no connection establishment and no connection termination as is the case for TCP. This means that each user datagram can travel on a different path.

One of the ramifications of being connectionless is that the process that uses UDP cannot send a stream of data to UDP and expect UDP to chop them into different related user datagrams. Instead each request must be small enough to fit into one user datagram. Only those processes sending short messages, messages less than 65,507 bytes (65,535 minus 8 bytes for the UDP header and minus 20 bytes for the IP header), can use UDP.

## Flow Control

UDP is a very simple protocol. There is no *flow control*, and hence no window mechanism. The receiver may overflow with incoming messages. The lack of flow control means that the process using UDP should provide for this service, if needed.

## Error Control

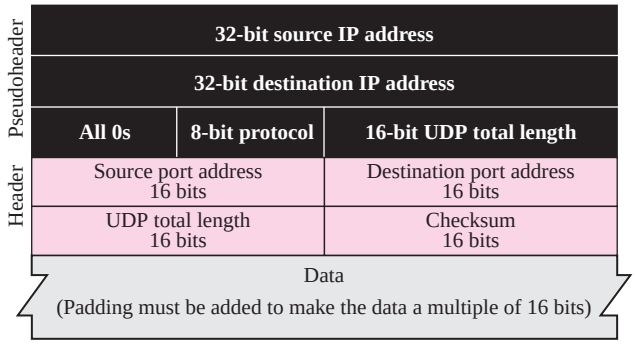
There is no *error control* mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded. The lack of error control means that the process using UDP should provide for this service if needed.

### *Checksum*

We have already talked about the concept of the *checksum* and the way it is calculated for IP in Chapter 7. UDP checksum calculation is different from the one for IP. Here the checksum includes three sections: a pseudoheader, the UDP header, and the data coming from the application layer.

The **pseudoheader** is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s (see Figure 14.3).

**Figure 14.3** *Pseudoheader for checksum calculation*




If the checksum does not include the pseudoheader, a user datagram may arrive safe and sound. However, if the IP header is corrupted, it may be delivered to the wrong host.

The protocol field is added to ensure that the packet belongs to UDP, and not to TCP. We will see later that if a process can use either UDP or TCP, the destination port number can be the same. The value of the protocol field for UDP is 17. If this value is changed during transmission, the checksum calculation at the receiver will detect it and UDP drops the packet. It is not delivered to the wrong protocol.

Note the similarities between the pseudoheader fields and the last 12 bytes of the IP header.

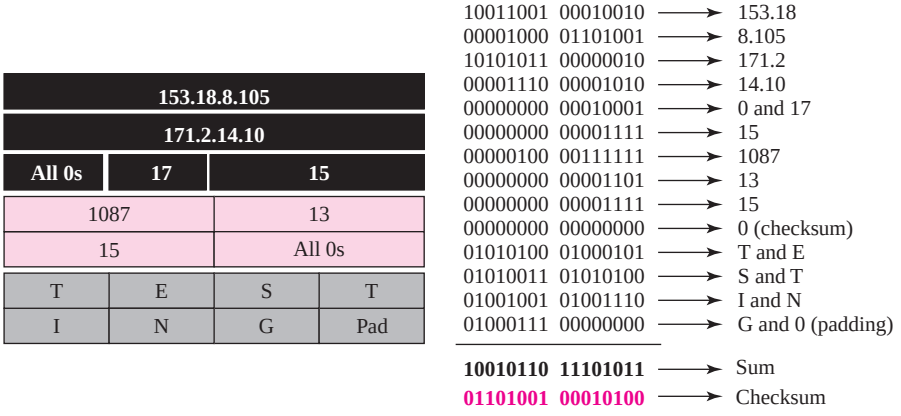
### Example 14.2

Figure 14.4 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calculation. The pseudoheader as well as the padding will be dropped when the user datagram is delivered to IP (see Appendix F).

### *Optional Inclusion of Checksum*

The sender of a UDP packet can choose not to calculate the checksum. In this case, the checksum field is filled with all 0s before being sent. In the situation that the sender decides to calculate the checksum, but it happens that the result is all 0s, the checksum is changed to all 1s before the packet is sent. In other words, the sender complements the sum two times. Note that this does not create confusion because the value of checksum is never all 1s in a normal situation (see the next example).

**Figure 14.4** *Checksum calculation of a simple UDP user datagram*




### Example 14.3

What value is sent for the checksum in one of the following hypothetical situations?

- **a.** The sender decides not to include the checksum.
- **b.** The sender decides to include the checksum, but the value of the sum is all 1s.
- **c.** The sender decides to include the checksum, but the value of the sum is all 0s.

**Solution**

- **a.** The value sent for the checksum field is all 0s to show that the checksum is not calculated.
- **b.** When the sender complements the sum, the result is all 0s; the sender complements the result again before sending. The value sent for the checksum is all 1s. The second complement operation is needed to avoid confusion with the case in part a.
- **c.** This situation never happens because it implies that the value of every term included in the calculation of the sum is all 0s, which is impossible; some fields in the pseudoheader have nonzero values (see Appendix D).

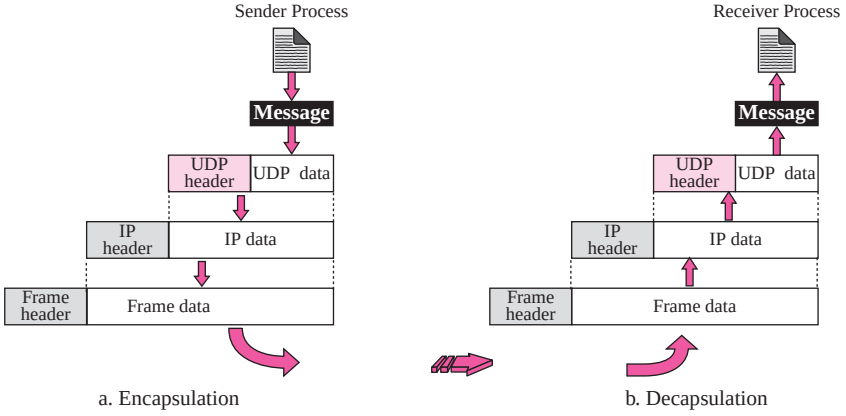
## Congestion Control

Since UDP is a connectionless protocol, it does not provide congestion control. UDP assumes that the packets sent are small and sporadic, and cannot create congestion in the network. This assumption may or may not be true today when UDP is used for real-time transfer of audio and video.

## Encapsulation and Decapsulation

To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages (see Figure 14.5).

**Figure 14.5** *Encapsulation and decapsulation*




### *Encapsulation*

When a process has a message to send through UDP, it passes the message to UDP along with a pair of socket addresses and the length of data. UDP receives the data and adds the UDP header. UDP then passes the user datagram to IP with the socket addresses. IP adds its own header, using the value 17 in the protocol field, indicating that the data has come from the UDP protocol. The IP datagram is then passed to the data link layer. The data link layer receives the IP datagram, adds its own header (and possibly a trailer), and passes it to the physical layer. The physical layer encodes the bits into electrical or optical signals and sends it to the remote machine.

### *Decapsulation*

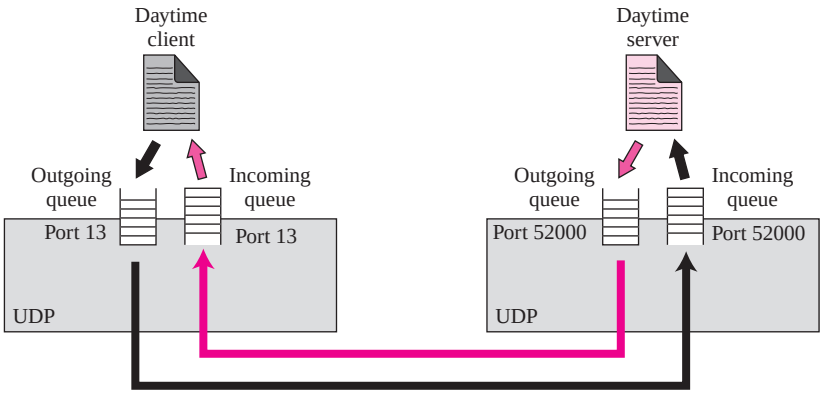
When the message arrives at the destination host, the physical layer decodes the signals into bits and passes it to the data link layer. The data link layer uses the header (and the trailer) to check the data. If there is no error, the header and trailer are dropped and the datagram is passed to IP. The IP software does its own checking. If there is no error, the header is dropped and the user datagram is passed to UDP with the sender and receiver IP addresses. UDP uses the checksum to check the entire user datagram. If there is no error, the header is dropped and the application data along with the sender socket address is passed to the process. The sender socket address is passed to the process in case it needs to respond to the message received.

## Queuing

We have talked about ports without discussing the actual implementation of them. In UDP, queues are associated with ports (see Figure 14.6).

At the client site, when a process starts, it requests a port number from the operating system. Some implementations create both an incoming and an outgoing queue associated with each process. Other implementations create only an incoming queue associated with each process.

**Figure 14.6** *Queues in UDP*




Note that even if a process wants to communicate with multiple processes, it obtains only one port number and eventually one outgoing and one incoming **queue.** The queues opened by the client are, in most cases, identified by ephemeral port numbers. The queues function as long as the process is running. When the process terminates, the queues are destroyed.

The client process can send messages to the outgoing queue by using the source port number specified in the request. UDP removes the messages one by one, and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow. If this happens, the operating system can ask the client process to wait before sending any more messages.

When a message arrives for a client, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field of the user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. If there is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a *port unreachable* message to the server. All of the incoming messages for one particular client program, whether coming from the same or a different server, are sent to the same queue. An incoming queue can overflow. If this happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the server.

At the server site, the mechanism of creating queues is different. In its simplest form, a server asks for incoming and outgoing queues using its well-known port when it starts running. The queues remain open as long as the server is running.

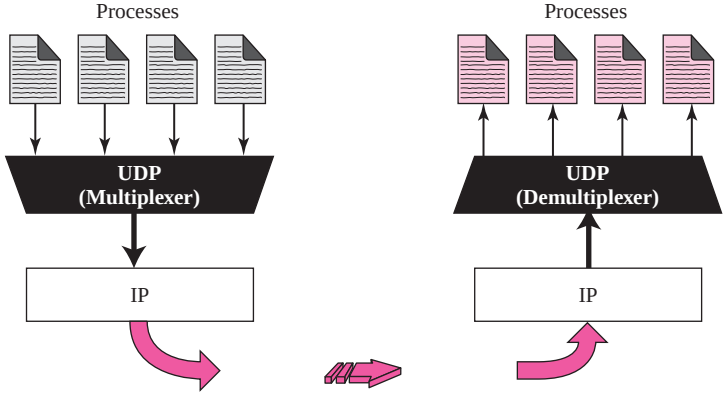
When a message arrives for a server, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field of the user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. If there is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a port unreachable message to the client. All of the incoming messages for one particular server, whether coming from the same or a different client, are sent to the same queue. An incoming queue can overflow. If this happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the client.

When a server wants to respond to a client, it sends messages to the outgoing queue using the source port number specified in the request. UDP removes the messages one by one, and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow. If this happens, the operating system asks the server to wait before sending any more messages.

## Multiplexing and Demultiplexing

In a host running a TCP/IP protocol suite, there is only one UDP but possibly several processes that may want to use the services of UDP. To handle this situation, UDP multiplexes and demultiplexes (see Figure 14.7).

**Figure 14.7** *Multiplexing and demultiplexing*




### *Multiplexing*

At the sender site, there may be several processes that need to send user datagrams. However, there is only one UDP. This is a many-to-one relationship and requires multiplexing. UDP accepts messages from different processes, differentiated by their assigned port numbers. After adding the header, UDP passes the user datagram to IP.

### *Demultiplexing*

At the receiver site, there is only one UDP. However, we may have many processes that can receive user datagrams. This is a one-to-many relationship and requires demultiplexing. UDP receives user datagrams from IP. After error checking and dropping of the header, UDP delivers each message to the appropriate process based on the port numbers.

## Comparison between UDP and Generic Simple Protocol

We can compare UDP with the connectionless simple protocol we discussed in Chapter 13. The only difference is that UDP provides an optional checksum to detect corrupted packets at the receiver site. If the checksum is added to the packet, the receiving UDP

can check the packet and discard the packet if it is corrupted. No feedback, however, is sent to the sender.

**UDP is an example of the connectionless simple protocol we discussed in Chapter 13 with the exception of an optional checksum added to packets for error detection.**

## 14.4 UDP APPLICATIONS

Although UDP meets almost none of the criteria we mentioned in Chapter 13 for a reliable transport-layer protocol, UDP is preferable for some applications. The reason is that some services may have some side effects that are either unacceptable or not preferable. An application designer needs sometimes to compromise to get the optimum. For example, in our daily life, we all know that a one-day delivery of a package by a carrier is more expensive than a three-day delivery. Although time and cost are both desirable features in delivery of a parcel, they are in conflict with each other. We need to choose the optimum.

In this section, we first discuss some features of UDP that may need to be considered when one designs an application program and then show some typical applications.

### UDP Features

We briefly discuss some features of UDP and their advantages and disadvantages.

#### *Connectionless Service*

As we mentioned previously, UDP is a connectionless protocol. Each UDP packet is independent from other packets sent by the same application program. This feature can be considered as an advantage or disadvantage depending on the application requirement. It is an advantage if, for example, a client application needs to send a short request to a server and to receive a short response. If the request and response can each fit in one single user datagram, a connectionless service may be preferable. The overhead to establish and close a connection may be significant in this case. In the connection-oriented service, to achieve the above goal, at least 9 packets are exchanged between the client and the server; in connectionless service only two packets are exchanged. The connectionless service provides less delay; the connection-oriented service creates more delay. If delay is an important issue for the application, the connectionless service is preferred.

#### Example 14.4

A client-server application such as DNS (see Chapter 19) uses the services of UDP because a client needs to send a short request to a server and to receive a quick response from it. The request and response can each fit in one user datagram. Since only one message is exchanged in each direction, the connectionless feature is not an issue; the client or server does not worry that messages are delivered out of order.

### Example 14.5

A client-server application such as SMTP (see Chapter 23), which is used in electronic mail, cannot use the services of UDP because a user can send a long e-mail message, which may include multimedia (images, audio, or video). If the application uses UDP and the message does not fit in one single user datagram, the message must be split by the application into different user datagrams. Here the connectionless service may create problems. The user datagrams may arrive and be delivered to the receiver application out of order. The receiver application may not be able to reorder the pieces. This means the connectionless service has a disadvantage for an application program that sends long messages. In SMTP, when one sends a message, one does not expect to receive a response quickly (sometimes no response is required). This means that the extra delay inherent in connection-oriented service is not crucial for SMTP.

#### *Lack of Error Control*

UDP does not provide error control; it provides an unreliable service. Most applications expect reliable service from a transport-layer protocol. Although a reliable service is desirable, it may have some side effects that are not acceptable to some applications. When a transport layer provides reliable services, if a part of the message is lost or corrupted, it needs to be resent. This means that the receiving transport layer cannot deliver that part to the application immediately; there is an uneven delay between different parts of the message delivered to the application layer. Some applications by nature do not even notice these uneven delays, but for some they are very crucial.

### Example 14.6

Assume we are downloading a very large text file from the Internet. We definitely need to use a transport layer that provides reliable service. We don't want part of the file to be missing or corrupted when we open the file. The delay created between the delivery of the parts are not an overriding concern for us; we wait until the whole file is composed before looking at it. In this case, UDP is not a suitable transport layer.

### Example 14.7

Assume we are watching a real-time stream video on our computer. Such a program is considered a long file; it is divided into many small parts and broadcast in real time. The parts of the message are sent one after another. If the transport layer is supposed to resend a corrupted or lost frame, the synchronizing of the whole transmission may be lost. The viewer suddenly sees a blank screen and needs to wait until the second transmission arrives. This is not tolerable. However, if each small part of the screen is sent using one single user datagram, the receiving UDP can easily ignore the corrupted or lost packet and deliver the rest to the application program. That part of the screen is blank for a very short period of the time, which most viewers do not even notice. However, video cannot be viewed out of order, so streaming audio, video, and voice applications that run over UDP must reorder or drop frames that are out of sequence.

#### *Lack of Congestion Control*

UDP does not provide congestion control. However, UDP does not create additional traffic in an error-prone network. TCP may resend a packet several times and thus contribute to the creation of congestion or worsen a congested situation. Therefore, in some cases, lack of error control in UDP can be considered an advantage when congestion is a big issue.

### Typical Applications

The following shows some typical applications that can benefit more from the services of UDP than from those of TCP.

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control. It is not usually used for a process such as FTP that needs to send bulk data (see Chapter 21).
- UDP is suitable for a process with internal flow and error-control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) (see Chapter 21) process includes flow and error control. It can easily use UDP.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP (see Chapter 24).
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP) (see Chapter 11).
- UDP is normally used for real-time applications that cannot tolerate uneven delay between sections of a received message.

## 14.5 UDP PACKAGE

To show how UDP handles the sending and receiving of UDP packets, we present a simple version of the UDP package.

We can say that the UDP package involves five components: a control-block table, input queues, a control-block module, an input module, and an output module. Figure 14.8 shows these five components and their interactions.

### Control-Block Table

In our package, UDP has a control-block table to keep track of the open ports. Each entry in this table has a minimum of four fields: the state, which can be FREE or IN-USE, the process ID, the port number, and the corresponding queue number.

### Input Queues

Our UDP package uses a set of input queues, one for each process. In this design, we do not use output queues.

### Control-Block Module

The control-block module (Table 14.2) is responsible for the management of the control-block table. When a process starts, it asks for a port number from the operating system. The operating system assigns well-known port numbers to servers and ephemeral port numbers to clients. The process passes the process ID and the port number to the control-block module to create an entry in the table for the process. The module

**Figure 14.8** *UDP design*

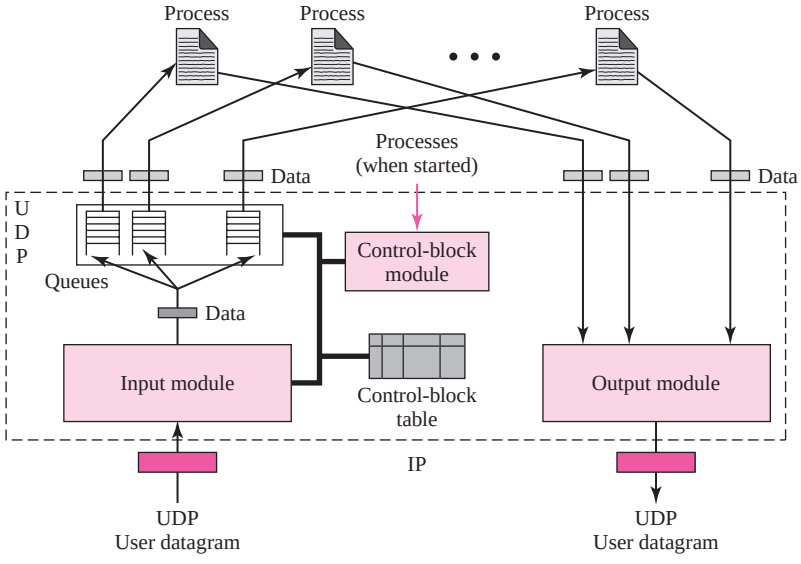



**Table 14.2** *Control Block Module*

```
1  UDP_Control_Block_Module (process ID, port number)
2  {
3       Search the table for a FREE entry.
4       if (not found)
5            Delete one entry using a predefined strategy.
6       Create a new entry with the state IN-USE
7       Enter the process ID and the port number.
8       Return.
9  } // End module
```

does not create the queues. The field for queue number has a value of zero. Note that we have not included a strategy to deal with a table that is full.

## Input Module

The input module (Table 14.3) receives a user datagram from the IP. It searches the control-block table to find an entry having the same port number as this user datagram. If the entry is found, the module uses the information in the entry to enqueue the data. If the entry is not found, it generates an ICMP message.

**Table 14.3** *Input Module*

```
1  UDP_INPUT_Module (user_datagram)
2  {
3       Look for the entry in the control_block table
4       if (found)
5       {
6            Check to see if a queue is allocated
7            If (queue is not allocated)
8                 allocate a queue
9            else
10                enqueue the data
11      } //end if
12      else
13      {
14           Ask ICMP to send an "unreachable port" message
15           Discard the user datagram
16      } //end else
17
18      Return.
19 } // end module
```

## Output Module

The output module (Table 14.4) is responsible for creating and sending user datagrams.

**Table 14.4** *Output Module*

```
1  UDP_OUTPUT_MODULE (Data)
2  {
3       Create a user datagram
4       Send the user datagram
5       Return.
6  }
```

## Examples

In this section we show some examples of how our package responds to input and output. The control-block table at the start of our examples is shown in Table 14.5.

**Table 14.5** *The Control-Block Table at the Beginning of Examples*

| State | Process ID | Port Number | Queue Number |
|---|---|---|---|
| IN-USE | 2,345 | 52,010 | 34 |
| IN-USE | 3,422 | 52,011 | |
| FREE | | | |
| IN-USE | 4,652 | 52,012 | 38 |
| FREE | | | |

### Example 14.8

The first activity is the arrival of a user datagram with destination port number 52,012. The input module searches for this port number and finds it. Queue number 38 has been assigned to this port, which means that the port has been previously used. The input module sends the data to queue 38. The control-block table does not change.

### Example 14.9

After a few seconds, a process starts. It asks the operating system for a port number and is granted port number 52,014. Now the process sends its ID (4,978) and the port number to the control-block module to create an entry in the table. The module takes the first FREE entry and inserts the information received. The module does not allocate a queue at this moment because no user datagrams have arrived for this destination (see Table 14.6).

**Table 14.6** *Control-Block Table after Example 14.9*

| State | Process ID | Port Number | Queue Number |
|---|---|---|---|
| IN-USE | 2,345 | 52,010 | 34 |
| IN-USE | 3,422 | 52,011 | |
| IN-USE | 4,978 | 52,014 | |
| IN-USE | 4,652 | 52,012 | 38 |
| FREE | | | |

### Example 14.10

A user datagram now arrives for port 52,011. The input module checks the table and finds that no queue has been allocated for this destination since this is the first time a user datagram has arrived for this destination. The module creates a queue and gives it a number (43). See Table 14.7.

**Table 14.7** *Control-Block Table after Example 14.10*

| State | Process ID | Port Number | Queue Number |
|---|---|---|---|
| IN-USE | 2,345 | 52,010 | 34 |
| IN-USE | 3,422 | 52,011 | 43 |
| IN-USE | 4,978 | 52,014 | |
| IN-USE | 4,652 | 52,012 | 38 |
| FREE | | | |

### Example 14.11

After a few seconds, a user datagram arrives for port 52,222. The input module checks the table and cannot find an entry for this destination. The user datagram is dropped and a request is made to ICMP to send an unreachable port message to the source.

## 14.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and websites. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give information about UDP. In particular, we recommend [Com 06] and [Ste 94].

### RFCs

The main RFC related to UDP is RFC 768.

## 14.7 KEY TERMS

connectionless, unreliable transport protocol
pseudoheader
queue
user datagram
User Datagram Protocol (UDP)

## 14.8 SUMMARY

- UDP is a transport protocol that creates a process-to-process communication. UDP is a (mostly) unreliable and connectionless protocol that requires little overhead and offers fast delivery. The UDP packet is called a user datagram.
- UDP's only attempt at error control is the checksum. Inclusion of a pseudoheader in the checksum calculation allows source and destination IP address errors to be detected. UDP has no flow-control mechanism.
- A user datagram is encapsulated in the data field of an IP datagram. Incoming and outgoing queues hold messages going to and from UDP.
- UDP uses multiplexing to handle outgoing user datagrams from multiple processes on one host. UDP uses demultiplexing to handle incoming user datagrams that go to different processes on the same host.
- A UDP package can involve five components: a control-block table, a control-block module, input queues, an input module, and an output module. The input queues hold incoming user datagrams. The control-block module is responsible for maintenance of entries in the control-block table. The input module creates input queues; the output module sends out user datagrams.

## 14.9 PRACTICE SET

### Exercises

1. In cases where reliability is not of primary importance, UDP would make a good transport protocol. Give examples of specific cases.
2. Are both UDP and IP unreliable to the same degree? Why or why not?
3. Show the entries for the header of a UDP user datagram that carries a message from a TFTP client to a TFTP server. Fill the checksum field with 0s. Choose an appropriate ephemeral port number and the correct well-known port number. The length of data is 40 bytes. Show the UDP packet using the format in Figure 14.2.
4. An SNMP client residing on a host with IP address 122.45.12.7 sends a message to an SNMP server residing on a host with IP address 200.112.45.90. What is the pair of sockets used in this communication?
5. A TFTP server residing on a host with IP address 130.45.12.7 sends a message to a TFTP client residing on a host with IP address 14.90.90.33. What is the pair of sockets used in this communication?
6. Answer the following questions:
   a. What is the minimum size of a UDP datagram?
   b. What is the maximum size of a UDP datagram?
   c. What is the minimum size of the process data that can be encapsulated in a UDP datagram?
   d. What is the maximum size of the process data that can be encapsulated in a UDP datagram?
7. A client uses UDP to send data to a server. The data length is 16 bytes. Calculate the efficiency of this transmission at the UDP level (ratio of useful bytes to total bytes).
8. Redo Exercise 7, calculating the efficiency of transmission at the IP level. Assume no options for the IP header.
9. Redo Exercise 7, calculating the efficiency of transmission at the data link layer. Assume no options for the IP header and use Ethernet at the data link layer.
10. The following is a dump of a UDP header in hexadecimal format.

**0045DF000058FE20**

   a. What is the source port number?
   b. What is the destination port number?
   c. What is the total length of the user datagram?
   d. What is the length of the data?
   e. Is the packet directed from a client to a server or vice versa?
   f. What is the client process?

CHAPTER

# 15

# *Transmission Control Protocol (TCP)*

As we discussed in Chapter 14, several protocols have been specified in the transport layer of the TCP/IP protocol suite. We will discuss TCP in this chapter. TCP is the heart of the suite, providing a vast array of services, and therefore, is a complicated protocol. TCP has gone through many revisions in the last few decades. This means that this chapter will be very long.

## OBJECTIVES

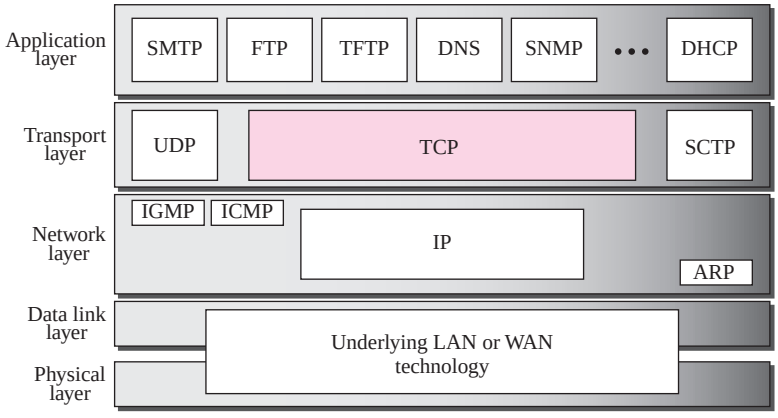
*The chapter has several objectives:*

- To introduce TCP as a protocol that provides reliable stream delivery service.
- To define TCP features and compare them with UDP features.
- To define the format of a TCP segment and its fields.
- To show how TCP provides a connection-oriented service, and show the segments exchanged during connection establishment and connection termination phases.
- To discuss the state transition diagram for TCP and discuss some scenarios.
- To introduce windows in TCP that are used for flow and error control.
- To discuss how TCP implements flow control in which the receive window controls the size of the send window.
- To discuss error control and FSMs used by TCP during the data transmission phase.
- To discuss how TCP controls the congestion in the network using different strategies.
- To list and explain the purpose of each timer in TCP.
- To discuss options in TCP and show how TCP can provide selective acknowledgment using the SACK option.
- To give a layout and a simplified pseudocode for the TCP package.

## 15.1 TCP SERVICES

Figure 15.1 shows the relationship of TCP to the other protocols in the TCP/IP protocol suite. TCP lies between the application layer and the network layer, and serves as the intermediary between the application programs and the network operations.

**Figure 15.1** *TCP/IP protocol suite*




Before discussing TCP in detail, let us explain the services offered by TCP to the processes at the application layer.

### Process-to-Process Communication

As with UDP, TCP provides process-to-process communication using port numbers (see Chapter 13). Table 15.1 lists some well-known port numbers used by TCP.

**Table 15.1** *Well-known Ports used by TCP*

| *Port* | *Protocol* | *Description* |
|---|---|---|
| 7 | Echo | Echoes a received datagram back to the sender |
| 9 | Discard | Discards any datagram that is received |
| 11 | Users | Active users |
| 13 | Daytime | Returns the date and the time |
| 17 | Quote | Returns a quote of the day |

**Table 15.1** *Well-known Ports used by TCP (continued)*

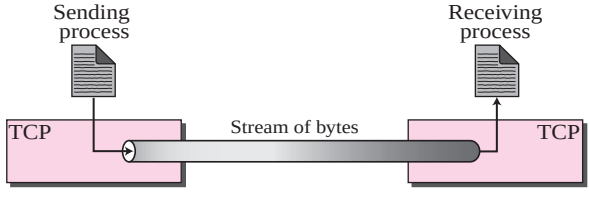
| *Port* | *Protocol* | *Description* |
|---|---|---|
| 19 | Chargen | Returns a string of characters |
| 20 and 21 | FTP | File Transfer Protocol (Data and Control) |
| 23 | TELNET | Terminal Network |
| 25 | SMTP | Simple Mail Transfer Protocol |
| 53 | DNS | Domain Name Server |
| 67 | BOOTP | Bootstrap Protocol |
| 79 | Finger | Finger |
| 80 | HTTP | Hypertext Transfer Protocol |

## Stream Delivery Service

TCP, unlike UDP, is a stream-oriented protocol. In UDP, a process sends messages with predefined boundaries to UDP for delivery. UDP adds its own header to each of these messages and delivers it to IP for transmission. Each message from the process is called a *user datagram*, and becomes, eventually, one IP datagram. Neither IP nor UDP recognizes any relationship between the datagrams.

TCP, on the other hand, allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their bytes across the Internet. This imaginary environment is depicted in Figure 15.2. The sending process produces (writes to) the stream of bytes and the receiving process consumes (reads from) them.

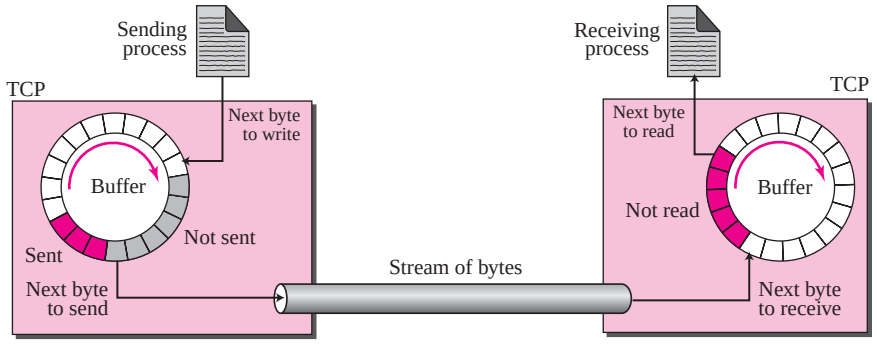
**Figure 15.2** *Stream delivery*




### *Sending and Receiving Buffers*

Because the sending and the receiving processes may not necessarily write or read data at the same rate, TCP needs buffers for storage. There are two buffers, the sending buffer and the receiving buffer, one for each direction. We will see later that these buffers are also necessary for flow- and error-control mechanisms used by TCP. One way to implement a buffer is to use a circular array of 1-byte locations as shown in Figure 15.3. For simplicity, we have shown two buffers of 20 bytes each; normally the buffers are hundreds or thousands of bytes, depending on the implementation. We also show the buffers as the same size, which is not always the case.

**Figure 15.3** *Sending and receiving buffers*




The figure shows the movement of the data in one direction. At the sender, the buffer has three types of chambers. The white section contains empty chambers that can be filled by the sending process (producer). The colored area holds bytes that have been sent but not yet acknowledged. The TCP sender keeps these bytes in the buffer until it receives an acknowledgment. The shaded area contains bytes to be sent by the sending TCP. However, as we will see later in this chapter, TCP may be able to send only part of this shaded section. This could be due to the slowness of the receiving process, or congestion in the network. Also note that after the bytes in the colored chambers are acknowledged, the chambers are recycled and available for use by the sending process. This is why we show a circular buffer.

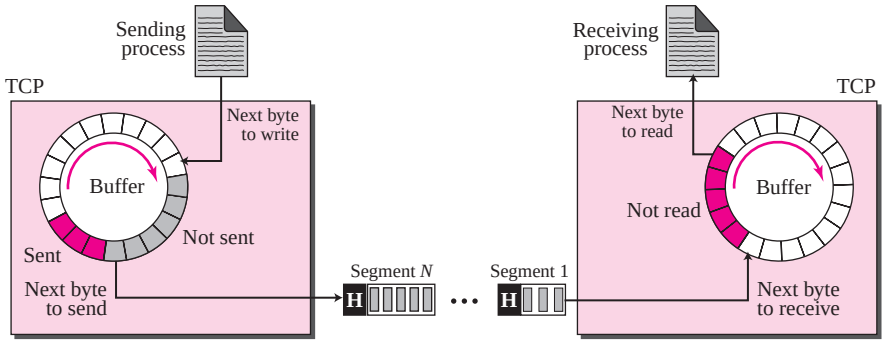
The operation of the buffer at the receiver is simpler. The circular buffer is divided into two areas (shown as white and colored). The white area contains empty chambers to be filled by bytes received from the network. The colored sections contain received bytes that can be read by the receiving process. When a byte is read by the receiving process, the chamber is recycled and added to the pool of empty chambers.

### *Segments*

Although buffering handles the disparity between the speed of the producing and consuming processes, we need one more step before we can send data. The IP layer, as a service provider for TCP, needs to send data in packets, not as a stream of bytes. At the transport layer, TCP groups a number of bytes together into a packet called a *segment*. TCP adds a header to each segment (for control purposes) and delivers the segment to the IP layer for transmission. The segments are encapsulated in an IP datagram and transmitted. This entire operation is transparent to the receiving process. Later we will see that segments may be received out of order, lost, or corrupted and resent. All of these are handled by the TCP sender with the receiving application process unaware of TCP's activities. Figure 15.4 shows how segments are created from the bytes in the buffers.

Note that segments are not necessarily all the same size. In the figure, for simplicity, we show one segment carrying 3 bytes and the other carrying 5 bytes. In reality, segments carry hundreds, if not thousands, of bytes.

**Figure 15.4** *TCP segments*




## Full-Duplex Communication

TCP offers *full-duplex service*, where data can flow in both directions at the same time. Each TCP endpoint then has its own sending and receiving buffer, and segments move in both directions.

## Multiplexing and Demultiplexing

Like UDP, TCP performs multiplexing at the sender and demultiplexing at the receiver. However, since TCP is a connection-oriented protocol, a connection needs to be established for each pair of processes. This will be more clear when we discuss the client/server paradigm in Chapter 17.

## Connection-Oriented Service

TCP, unlike UDP, is a connection-oriented protocol. As shown in Chapter 13, when a process at site A wants to send to and receive data from another process at site B, the following three phases occur:

1. The two TCPs establish a virtual connection between them.
2. Data are exchanged in both directions.
3. The connection is terminated.

Note that this is a virtual connection, not a physical connection. The TCP segment is encapsulated in an IP datagram and can be sent out of order, or lost, or corrupted, and then resent. Each may be routed over a different path to reach the destination. There is no physical connection. TCP creates a stream-oriented environment in which it accepts the responsibility of delivering the bytes in order to the other site.

## Reliable Service

TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data. We will discuss this feature further in the section on error control.

## 15.2 TCP FEATURES

To provide the services mentioned in the previous section, TCP has several features that are briefly summarized in this section and discussed later in detail.

### Numbering System

Although the TCP software keeps track of the segments being transmitted or received, there is no field for a segment number value in the segment header. Instead, there are two fields called the *sequence number* and the *acknowledgment number*. These two fields refer to a byte number and not a segment number.

#### *Byte Number*

TCP numbers all data bytes (octets) that are transmitted in a connection. Numbering is independent in each direction. When TCP receives bytes of data from a process, TCP stores them in the sending buffer and numbers them. The numbering does not necessarily start from 0. Instead, TCP chooses an arbitrary number between 0 and $2^{32} - 1$ for the number of the first byte. For example, if the number happens to be 1,057 and the total data to be sent is 6,000 bytes, the bytes are numbered from 1,057 to 7,056. We will see that byte numbering is used for flow and error control.

**The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with an arbitrarily generated number.**

#### *Sequence Number*

After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent. The sequence number for each segment is the number of the first byte of data carried in that segment.

#### Example 15.1

Suppose a TCP connection is transferring a file of 5,000 bytes. The first byte is numbered 10,001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1,000 bytes?

**Solution**

The following shows the sequence number for each segment:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Segment 1** | → | **Sequence Number:** | **10,001** | **Range:** | **10,001** | to | **11,000** |
| **Segment 2** | → | **Sequence Number:** | **11,001** | **Range:** | **11,001** | to | **12,000** |
| **Segment 3** | → | **Sequence Number:** | **12,001** | **Range:** | **12,001** | to | **13,000** |
| **Segment 4** | → | **Sequence Number:** | **13,001** | **Range:** | **13,001** | to | **14,000** |
| **Segment 5** | → | **Sequence Number:** | **14,001** | **Range:** | **14,001** | to | **15,000** |

**The value in the sequence number field of a segment defines the number assigned to the first data byte contained in that segment.**

When a segment carries a combination of data and control information (piggybacking), it uses a sequence number. If a segment does not carry user data, it does not logically define a sequence number. The field is there, but the value is not valid. However, some segments, when carrying only control information, need a sequence number to allow an acknowledgment from the receiver. These segments are used for connection establishment, termination, or abortion. Each of these segments consume one sequence number as though it carries one byte, but there are no actual data. We will elaborate on this issue when we discuss connections.

### *Acknowledgment Number*

As we discussed previously, communication in TCP is full duplex; when a connection is established, both parties can send and receive data at the same time. Each party numbers the bytes, usually with a different starting byte number. The sequence number in each direction shows the number of the first byte carried by the segment. Each party also uses an acknowledgment number to confirm the bytes it has received. However, the acknowledgment number defines the number of the next byte that the party expects to receive. In addition, the acknowledgment number is cumulative, which means that the party takes the number of the last byte that it has received, safe and sound, adds 1 to it, and announces this sum as the acknowledgment number. The term *cumulative* here means that if a party uses 5,643 as an acknowledgment number, it has received all bytes from the beginning up to 5,642. Note that this does not mean that the party has received 5,642 bytes because the first byte number does not have to start from 0.

**The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive. The acknowledgment number is cumulative.**

## Flow Control

TCP, unlike UDP, provides flow control. The sending TCP controls how much data can be accepted from the sending process; the receiving TCP controls how much data can to be sent by the sending TCP (See Chapter 13). This is done to prevent the receiver from being overwhelmed with data. The numbering system allows TCP to use a byte-oriented flow control, as we discuss later in the chapter.

## Error Control

To provide reliable service, TCP implements an error control mechanism. Although error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented, as we will see later.

### Congestion Control

TCP, unlike UDP, takes into account congestion in the network. The amount of data sent by a sender is not only controlled by the receiver (flow control), but is also determined by the level of congestion, if any, in the network.

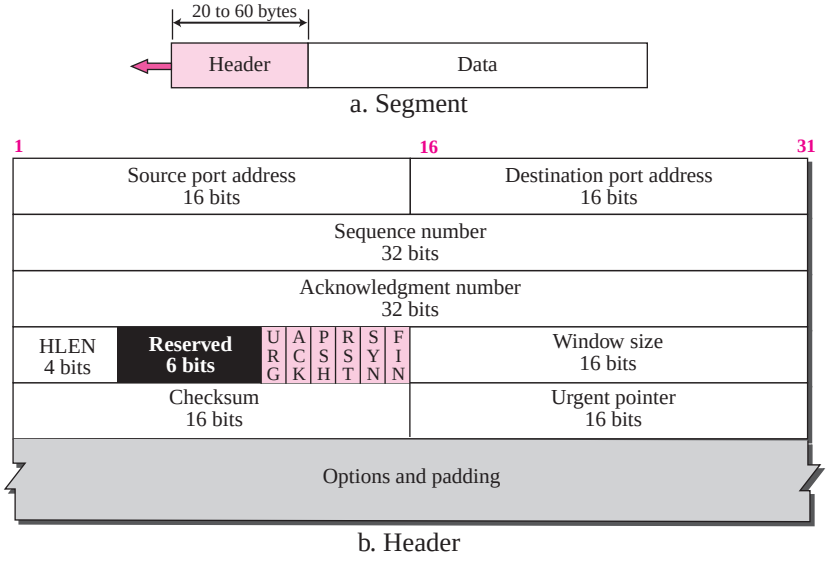
## 15.3 SEGMENT

Before discussing TCP in more detail, let us discuss the TCP packets themselves. A packet in TCP is called a **segment.**

### Format

The format of a segment is shown in Figure 15.5. The segment consists of a header of 20 to 60 bytes, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options. We will discuss some of the header fields in this section. The meaning and purpose of these will become clearer as we proceed through the chapter.

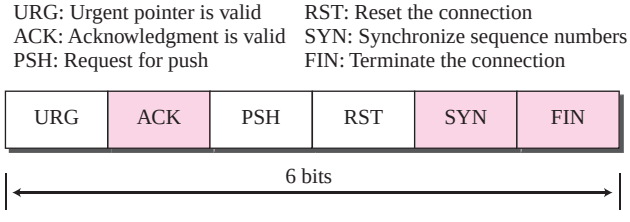
**Figure 15.5** *TCP segment format*




- **Source port address.** This is a 16-bit field that defines the port number of the application program in the host that is sending the segment. This serves the same purpose as the source port address in the UDP header discussed in Chapter 14.

- **Destination port address.** This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment. This serves the same purpose as the destination port address in the UDP header discussed in Chapter 14.
- **Sequence number.** This 32-bit field defines the number assigned to the first byte of data contained in this segment. As we said before, TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination which byte in this sequence is the first byte in the segment. During connection establishment (discussed later) each party uses a random number generator to create an **initial sequence number** (ISN), which is usually different in each direction.
- **Acknowledgment number.** This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number $x$ from the other party, it returns $x + 1$ as the acknowledgment number. Acknowledgment and data can be piggybacked together.
- **Header length.** This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field is always between 5 ($5 \times 4 = 20$) and 15 ($15 \times 4 = 60$).
- **Reserved.** This is a 6-bit field reserved for future use.
- **Control.** This field defines 6 different control bits or flags as shown in Figure 15.6. One or more of these bits can be set at a time. These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP. A brief description of each bit is shown in the figure. We will discuss them further when we study the detailed operation of TCP later in the chapter.

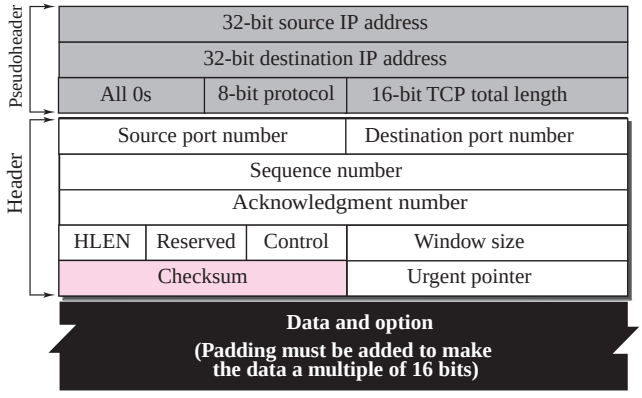
**Figure 15.6** *Control field*




- **Window size.** This field defines the window size of the sending TCP in bytes. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window (*rwnd*) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.
- **Checksum.** This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as the one described for UDP in Chapter 14.

However, the use of the checksum in the UDP datagram is optional, whereas the use of the checksum for TCP is mandatory. The same pseudoheader, serving the same purpose, is added to the segment. For the TCP pseudoheader, the value for the protocol field is 6. See Figure 15.7.

**Figure 15.7** *Pseudoheader added to the TCP datagram*



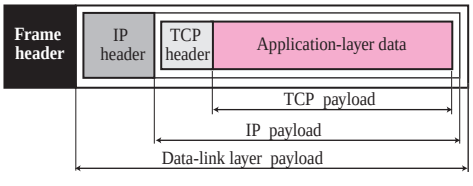

**The use of the checksum in TCP is mandatory.**

- **Urgent pointer.** This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines a value that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment. This will be discussed later in this chapter.
- **Options.** There can be up to 40 bytes of optional information in the TCP header. We will discuss the different options currently used in the TCP header later in the chapter.

## Encapsulation

A TCP segment encapsulates the data received from the application layer. The TCP segment is encapsulated in an IP datagram, which in turn is encapsulated in a frame at the data-link layer as shown in Figure 15.8.

**Figure 15.8** *Encapsulation*



## 15.4 A TCP CONNECTION

TCP is connection-oriented. As discussed in Chapter 13, a connection-oriented transport protocol establishes a virtual path between the source and destination. All of the segments belonging to a message are then sent over this virtual path. Using a single virtual pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames. You may wonder how TCP, which uses the services of IP, a connectionless protocol, can be connection-oriented. The point is that a TCP connection is virtual, not physical. TCP operates at a higher level. TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself. If a segment is lost or corrupted, it is retransmitted. Unlike TCP, IP is unaware of this retransmission. If a segment arrives out of order, TCP holds it until the missing segments arrive; IP is unaware of this reordering.

In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.

### Connection Establishment

TCP transmits data in full-duplex mode. When two TCPs in two machines are connected, they are able to send segments to each other simultaneously. This implies that each party must initialize communication and get approval from the other party before any data are transferred.

#### *Three-Way Handshaking*

The connection establishment in TCP is called **three-way handshaking.** In our example, an application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport layer protocol.

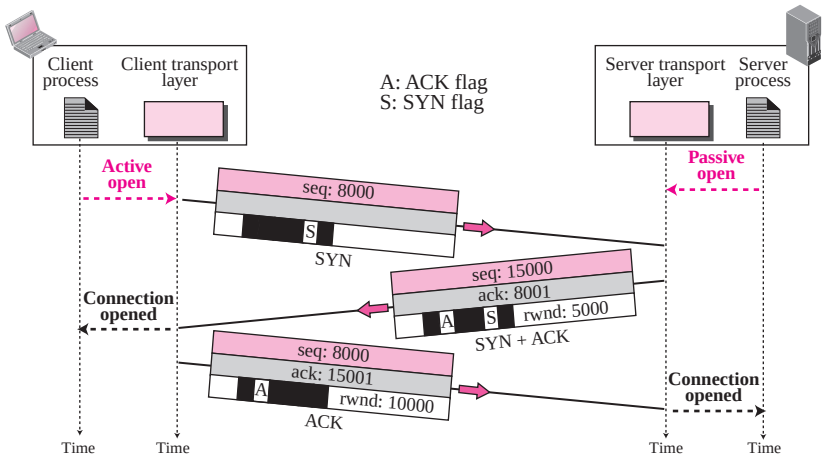
The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This request is called a *passive open*. Although the server TCP is ready to accept a connection from any machine in the world, it cannot make the connection itself.

The client program issues a request for an *active open*. A client that wishes to connect to an open server tells its TCP to connect to a particular server. TCP can now start the three-way handshaking process as shown in Figure 15.9.

To show the process we use time lines. Each segment has values for all its header fields and perhaps for some of its option fields too. However, we show only the few fields necessary to understand each phase. We show the sequence number, the acknowledgment number, the control flags (only those that are set), and window size if relevant. The three steps in this phase are as follows.

1. The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for synchronization of sequence numbers. The client in our example chooses a random number as the first sequence number and sends this number to the server. This sequence number is called the initial sequence number (ISN). Note that this segment does not contain an acknowledgment number. It does not define the window size either; a window size definition makes sense only when a segment includes an acknowledgment. The segment can also include some

**Figure 15.9** *Connection establishment using three-way handshaking*




options that we discuss later in the chapter. Note that the SYN segment is a control segment and carries no data. However, it consumes one sequence number. When the data transfer starts, the ISN is incremented by 1. We can say that the SYN segment carries no real data, but we can think of it as containing one imaginary byte.

**A SYN segment cannot carry data, but it consumes one sequence number.**

2. The server sends the second segment, a SYN + ACK segment with two flag bits set: SYN and ACK. This segment has a dual purpose. First, it is a SYN segment for communication in the other direction. The server uses this segment to initialize a sequence number for numbering the bytes sent from the server to the client. The server also acknowledges the receipt of the SYN segment from the client by setting the ACK flag and displaying the next sequence number it expects to receive from the client. Because it contains an acknowledgment, it also needs to define the receive window size, *rwnd* (to be used by the client), as we will see in the flow control section.

**A SYN + ACK segment cannot carry data, but does consume one sequence number.**

3. The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field. Note that the sequence number in this segment is the same as the one in the SYN segment; the ACK segment does not consume any sequence numbers. The client must also define the server window size. Some implementations allow this third segment in the connection phase to carry the first chunk of data from the

client. In this case, the third segment must have a new sequence number showing the byte number of the first byte in the data. In general, the third segment usually does not carry data and consumes no sequence numbers.

**An ACK segment, if carrying no data, consumes no sequence number.**

### *Simultaneous Open*

A rare situation may occur when both processes issue an active open. In this case, both TCPs transmit a SYN + ACK segment to each other and one single connection is established between them. We will show this case when we discuss the transition diagram in the next section.

### *SYN Flooding Attack*

The connection establishment procedure in TCP is susceptible to a serious security problem called **SYN flooding attack.** This happens when one or more malicious attackers send a large number of SYN segments to a server pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams. The server, assuming that the clients are issuing an active open, allocates the necessary resources, such as creating transfer control block (TCB) tables (explained later in the chapter) and setting timers. The TCP server then sends the SYN + ACK segments to the fake clients, which are lost. When the server waits for the third leg of the handshaking process, however, resources are allocated without being used. If, during this short period of time, the number of SYN segments is large, the server eventually runs out of resources and may be unable to accept connection requests from valid clients. This SYN flooding attack belongs to a group of security attacks known as a **denial of service attack,** in which an attacker monopolizes a system with so many service requests that the system overloads and denies service to valid requests.

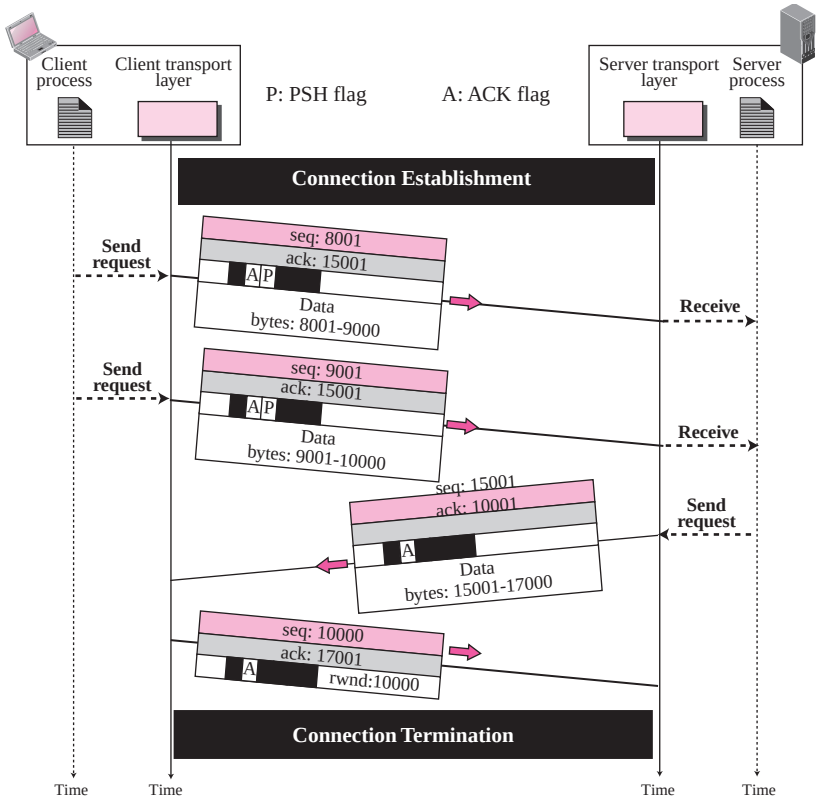
Some implementations of TCP have strategies to alleviate the effect of a SYN attack. Some have imposed a limit of connection requests during a specified period of time. Others try to filter out datagrams coming from unwanted source addresses. One recent strategy is to postpone resource allocation until the server can verify that the connection request is coming from a valid IP address, by using what is called a **cookie.** SCTP, the new transport-layer protocol that we discuss in the next chapter, uses this strategy.

## Data Transfer

After connection is established, bidirectional **data transfer** can take place. The client and server can send data and acknowledgments in both directions. We will study the rules of acknowledgment later in the chapter; for the moment, it is enough to know that data traveling in the same direction as an acknowledgment are carried on the same segment. The acknowledgment is piggybacked with the data. Figure 15.10 shows an example.

In this example, after a connection is established, the client sends 2,000 bytes of data in two segments. The server then sends 2,000 bytes in one segment. The client sends one more segment. The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there is no more data to be sent. Note the values of the sequence and acknowledgment numbers. The data segments sent by the client have the PSH (push) flag set so that the server TCP tries to deliver data to the server process as soon as they are received. We discuss the use of this

**Figure 15.10** *Data transfer*




flag in more detail later. The segment from the server, on the other hand, does not set the push flag. Most TCP implementations have the option to set or not set this flag.

### *Pushing Data*

We saw that the sending TCP uses a buffer to store the stream of data coming from the sending application program. The sending TCP can select the segment size. The receiving TCP also buffers the data when they arrive and delivers them to the application program when the application program is ready or when it is convenient for the receiving TCP. This type of flexibility increases the efficiency of TCP.

However, there are occasions in which the application program has no need for this flexibility. For example, consider an application program that communicates interactively with another application program on the other end. The application program on one site wants to send a keystroke to the application at the other site and receive an immediate response. Delayed transmission and delayed delivery of data may not be acceptable by the application program.

TCP can handle such a situation. The application program at the sender can request a *push* operation. This means that the sending TCP must not wait for the window to be filled. It must create a segment and send it immediately. The sending TCP must also set

the push bit (PSH) to let the receiving TCP know that the segment includes data that must be delivered to the receiving application program as soon as possible and not to wait for more data to come.

Although the push operation can be requested by the application program, most current TCP implementations ignore such requests. TCP can choose whether or not to use this feature.

### *Urgent Data*

TCP is a stream-oriented protocol. This means that the data is presented from the application program to TCP as a stream of bytes. Each byte of data has a position in the stream. However, there are occasions in which an application program needs to send *urgent* bytes, some bytes that need to be treated in a special way by the application at the other end. The solution is to send a segment with the URG bit set. The sending application program tells the sending TCP that the piece of data is urgent. The sending TCP creates a segment and inserts the urgent data at the beginning of the segment. The rest of the segment can contain normal data from the buffer. The urgent pointer field in the header defines the end of the urgent data (the last byte of urgent data).

When the receiving TCP receives a segment with the URG bit set, it informs the receiving application of the situation. How this is done, depends on the operation system. It is then to the discretion of the receiving program to take an action.

It is important to mention that TCP's urgent data is neither a priority service nor an expedited data service. Rather, TCP urgent mode is a service by which the application program at the sender side marks some portion of the byte stream as needing special treatment by the application program at the receiver side.

Thus, signaling the presence of urgent data and marking its position in the data stream are the only aspects that distinguish the delivery of urgent data from the delivery of all other TCP data. For all other purposes, urgent data is treated identically to the rest of the TCP byte stream. The application program at the receiver site must read every byte of data exactly in the order it was submitted regardless of whether or not urgent mode is used. The standard TCP, as implemented, does not ever deliver any data out of order.

## Connection Termination

Any of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client. Most implementations today allow two options for connection termination: three-way handshaking and four-way handshaking with a half-close option.

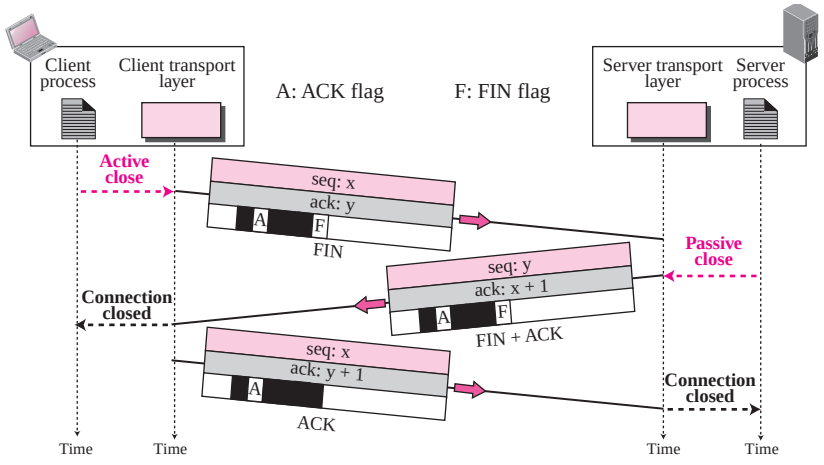
### *Three-Way Handshaking*

Most implementations today allow *three-way handshaking* for connection termination as shown in Figure 15.11.

1. In a common situation, the client TCP, after receiving a close command from the client process, sends the first segment, a FIN segment in which the FIN flag is set. Note that a FIN segment can include the last chunk of data sent by the client or it can be just a control segment as shown in the figure. If it is only a control segment, it consumes only one sequence number.

**The FIN segment consumes one sequence number if it does not carry data.**

**Figure 15.11** *Connection termination using three-way handshaking*




2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN+ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction. This segment can also contain the last chunk of data from the server. If it does not carry data, it consumes only one sequence number.
3. The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server. This segment contains the acknowledgment number, which is one plus the sequence number received in the FIN segment from the server. This segment cannot carry data and consumes no sequence numbers.

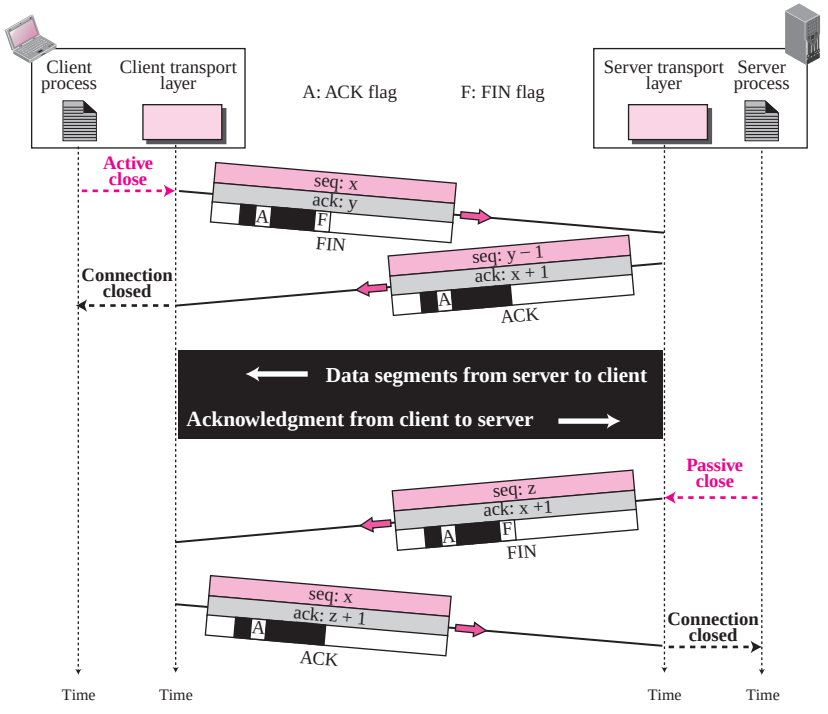
### *Half-Close*

In TCP, one end can stop sending data while still receiving data. This is called a **half-close.** Either the server or the client can issue a half-close request. It can occur when the server needs all the data before processing can begin. A good example is sorting. When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start. This means the client, after sending all data, can close the connection in the client-to-server direction. However, the server-to-client direction must remain open to return the sorted data. The server, after receiving the data, still needs time for sorting; its outbound direction must remain open.

**The FIN + ACK segment consumes one sequence number if it does not carry data.**

Figure 15.12 shows an example of a half-close. The data transfer from the client to the server stops. The client half-closes the connection by sending a FIN segment. The server accepts the half-close by sending the ACK segment. The server, however, can still send data. When the server has sent all of the processed data, it sends a FIN segment, which is acknowledged by an ACK from the client.

**Figure 15.12** *Half-close*




After half closing the connection, data can travel from the server to the client and acknowledgments can travel from the client to the server. The client cannot send any more data to the server. Note the sequence numbers we have used. The second segment (ACK) consumes no sequence number. Although the client has received sequence number $y - 1$ and is expecting $y$, the server sequence number is still $y - 1$. When the connection finally closes, the sequence number of the last ACK segment is still $x$, because no sequence numbers are consumed during data transfer in that direction.

## Connection Reset

TCP at one end may deny a connection request, may abort an existing connection, or may terminate an idle connection. All of these are done with the RST (reset) flag.

### *Denying a Connection*

Suppose the TCP on one side has requested a connection to a nonexistent port. The TCP on the other side may send a segment with its RST bit set to deny the request. We will show an example of this case in the next section.

### *Aborting a Connection*

One TCP may want to abort an existing connection due to an abnormal situation. It can send an RST segment to close the connection. We also show an example of this case in the next section.

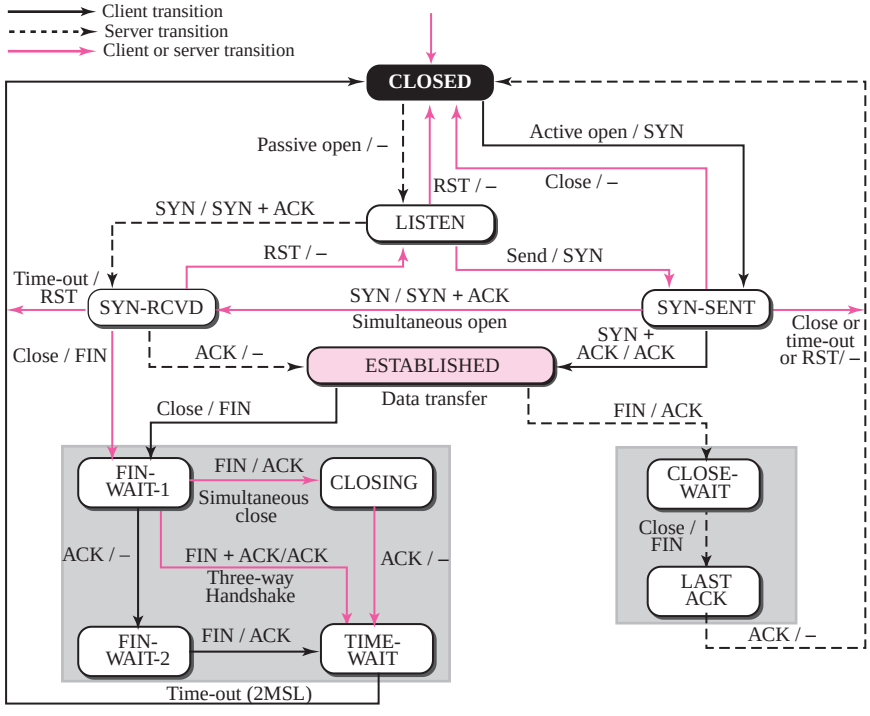
### *Terminating an Idle Connection*

The TCP on one side may discover that the TCP on the other side has been idle for a long time. It may send an RST segment to end the connection. The process is the same as aborting a connection.

## 15.5 STATE TRANSITION DIAGRAM

To keep track of all the different events happening during connection establishment, connection termination, and data transfer, TCP is specified as the finite state machine shown in Figure 15.13.

**Figure 15.13** *State transition diagram*




The figure shows the two FSMs used by the TCP client and server combined in one diagram. The ovals represent the states. The transition from one state to another is shown using directed lines. Each line has two strings separated by a slash. The first string is the input, what TCP receives. The second is the output, what TCP sends. The dotted black lines in the figure represent the transition that a server normally goes through; the solid black lines show the transitions that a client normally goes through. However, in some situations, a server transitions through a solid line or a client transitions through a dotted line. The colored lines show special situations. Note that the oval marked as ESTABLISHED is

in fact two sets of states, a set for the client and another for the server, that are used for flow and error control as explained later in the chapter. We use several scenarios based on Figure 15.13 and show the part of the figure in each case.

**The state marked as ESTBLISHED in the FSM is in fact two different sets of states that the client and server undergo to transfer data.**

Table 15.2 shows the list of states for TCP.

**Table 15.2** *States for TCP*

| *State* | *Description* |
|---|---|
| **CLOSED** | No connection exists |
| **LISTEN** | Passive open received; waiting for SYN |
| **SYN-SENT** | SYN sent; waiting for ACK |
| **SYN-RCVD** | SYN+ACK sent; waiting for ACK |
| **ESTABLISHED** | Connection established; data transfer in progress |
| **FIN-WAIT-1** | First FIN sent; waiting for ACK |
| **FIN-WAIT-2** | ACK to first FIN received; waiting for second FIN |
| **CLOSE-WAIT** | First FIN received, ACK sent; waiting for application to close |
| **TIME-WAIT** | Second FIN received, ACK sent; waiting for 2MSL time-out |
| **LAST-ACK** | Second FIN sent; waiting for ACK |
| **CLOSING** | Both sides decided to close simultaneously |

## Scenarios

To understand the TCP state machines and the transition diagrams, we go through some scenarios in this section.

### *Connection Establishment and Half-Close Termination*

We show a scenario where the server process issues a passive open and passive close, and the client process issues an active open and active close. The half-close termination allows us to show more states. Figure 15.14 shows two state transition diagrams for the client and server.

Figure 15.15 shows the same idea using a time-line diagram.

**Client States** The client process issues a command to its TCP to request a connection to a specific socket address. This called an *active open*. TCP sends a SYN segment and moves to the **SYN-SENT** state. After receiving the SYN+ACK segment, TCP sends an ACK segment and goes to the **ESTABLISHED** state. Data are transferred, possibly in both directions, and acknowledged. When the client process has no more data to send, it issues a command called an *active close*. The client TCP sends a FIN segment and goes to the **FIN-WAIT-1** state. When it receives the ACK for the sent FIN, it goes to **FIN-WAIT-2** state and remains there until it receives a FIN segment from the

**Figure 15.14** *Transition diagrams for connection establishment and half-close termination*

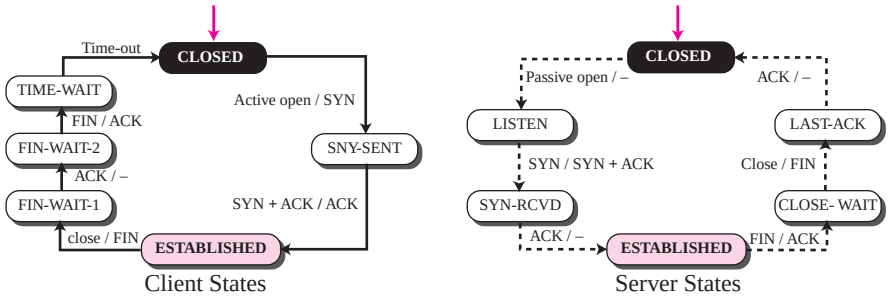
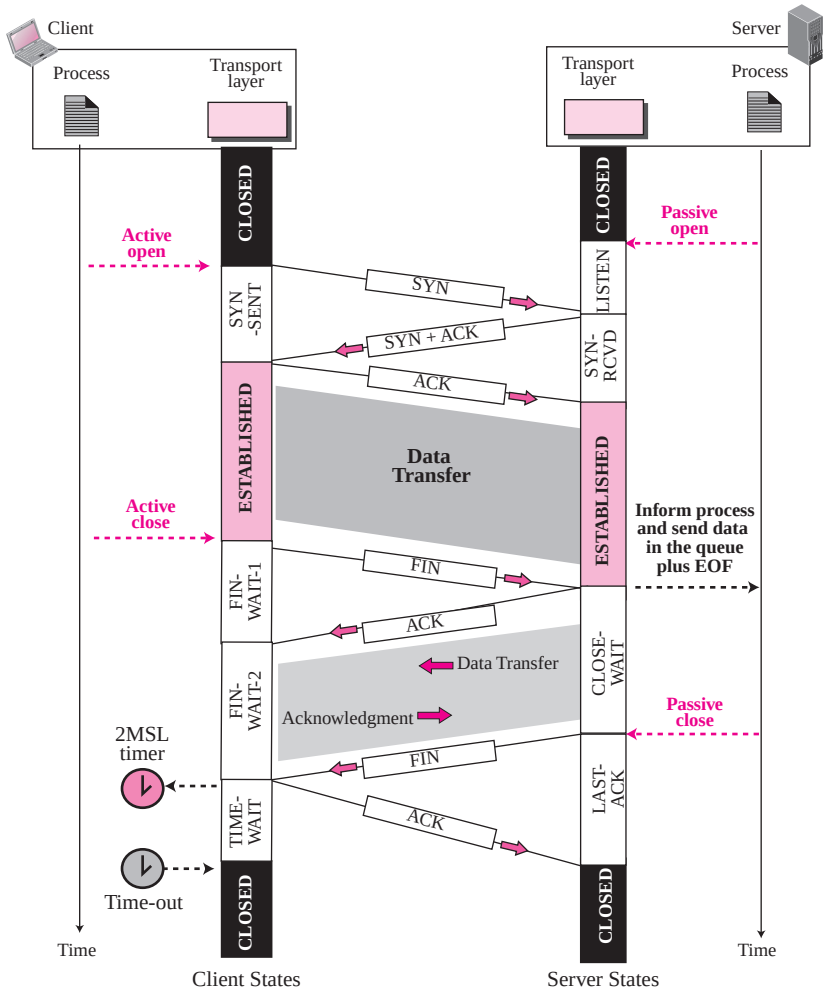



**Figure 15.15** *Time-line diagram for connection establishment and half-close termination*



server. When the FIN segment is received, the client sends an ACK segment and goes to the **TIME-WAIT** state and sets a timer for a time-out value of twice the maximum segment lifetime (MSL). The MSL is the maximum time a segment can exist in the Internet before it is dropped. Remember that a TCP segment is encapsulated in an IP datagram, which has a limited lifetime (TTL). When the IP datagram is dropped, the encapsulated TCP segment is also dropped. The common value for MSL is between 30 seconds and 1 minute. There are two reasons for the existence of the **TIME-WAIT** state and the 2SML timer:

1. If the last ACK segment is lost, the server TCP, which sets a timer for the last FIN, assumes that its FIN is lost and resends it. If the client goes to the **CLOSED** state and closes the connection before the 2MSL timer expires, it never receives this resent FIN segment, and consequently, the server never receives the final ACK. The server cannot close the connection. The 2MSL timer makes the client wait for a duration that is enough time for an ACK to be lost (one SML) and a FIN to arrive (another SML). If during the **TIME-WAIT** state, a new FIN arrives, the client sends a new ACK and restarts the 2SML timer.
2. A duplicate segment from one connection might appear in the next one. Assume a client and a server have closed a connection. After a short period of time, they open a connection with the same socket addresses (same source and destination IP addresses and same source and destination port numbers). This new connection is called an *incarnation* of the old one. A duplicated segment from the previous connection may arrive in this new connection and be interpreted as belonging to the new connection if there is not enough time between the two connections. To prevent this problem, TCP requires that an incarnation cannot occur unless 2MSL amount of time has elapsed. Some implementations, however, ignore this rule if the initial sequence number of the incarnation is greater than the last sequence number used in the previous connection.

**Server States** In our scenario, the server process issues an *open* command. This must happen before the client issues an *open* command. The server TCP goes to the **LISTEN** state and remains there, passively, until it receives a SYN segment. When the server TCP receives a SYN segment, it sends a SYN+ACK segment and goes to **SYN-RCVD** state, waiting for the client to send an ACK segment. After receiving the ACK segment, it goes to **ESTABLISHED** state, where data transfer can take place.

Note that although either side (client or server) may initiate the close, we assume that the client initiates the close without loss of generality.

TCP remains in this state until it receives a FIN segment from the client TCP signifying that there are no more data to be sent and that the connection can be closed. At this moment, the server sends an ACK to the client, delivers outstanding data in its queue to the application, and goes to the **CLOSE-WAIT** state. In our scenario, we assume a half-close connection. The server TCP can still send data to the client and receive acknowledgments, but no data can flow in the other direction. The server TCP remains in this state until the application actually issues a *close* command. It then sends a FIN to the client to show that it is closing the connection too, and goes to **LAST-ACK** state. It remains in this state until it receives the final ACK, when it then goes to the **CLOSED** state. The termination phase beginning with the first FIN is called a four-way handshake.

### *A Common Scenario*

As mentioned before, three-way handshake in the connection establishment and connection terminations phases are common. Figure 15.16 shows the state transition diagram for the client and server in this scenario.

**Figure 15.16** *Transition diagram for a common scenario*

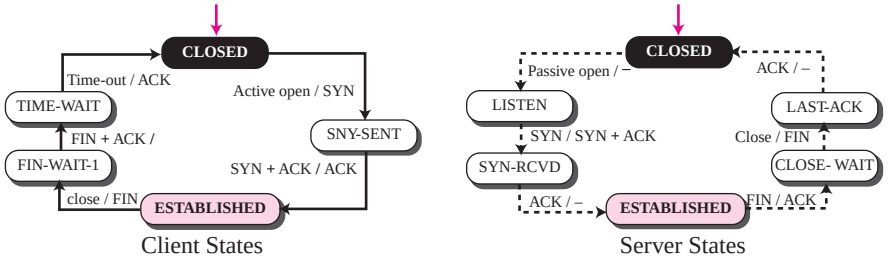
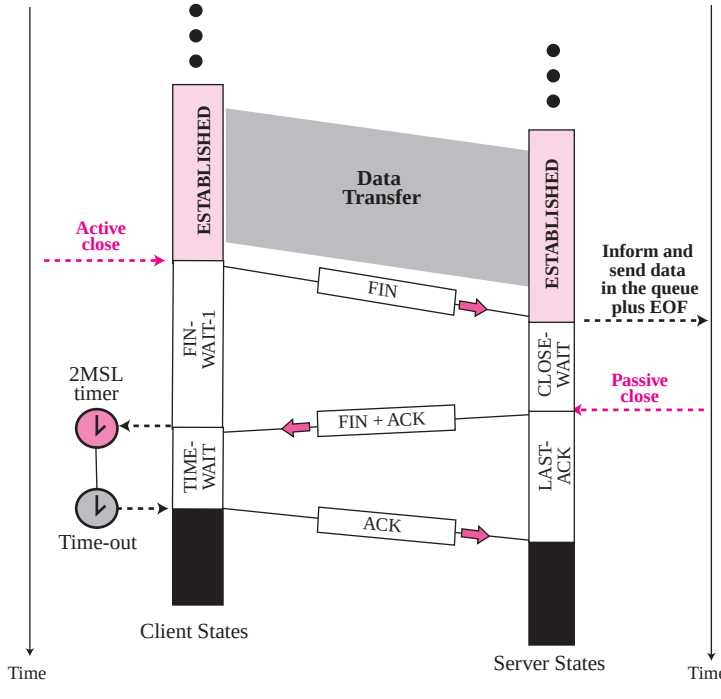



Figure 15.17 shows the same scenario with states over the time line. The connection establishment phase is the same as the one in the previous scenario; we show only the connection termination phase.

**Figure 15.17** *Time-line diagram for a common scenario*

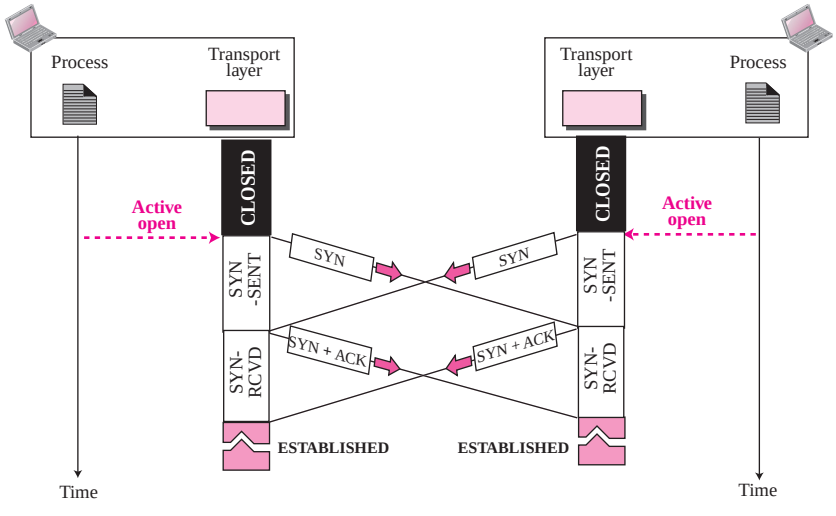


The figure shows that the client issues a *close* after the data transfer phase. The client TCP sends a FIN segment and goes to **FIN-WAIT-1** state. The server TCP, upon receiving the FIN segment, sends all queued data to the server with a virtual EOF marker, which means that the connection must be closed. It goes to the **CLOSE-WAIT** state, but postpones acknowledging the FIN segment received from the client until it receives a passive close from its process. After receiving the passive close command, the server sends a FIN+ACK segment to the client and goes to the **LAST-ACK** state, waiting for the final ACK. The client eliminates the **FIN-WAIT-2** state and goes directly to the **TIME-WAIT** state. The rest is the same as four-way handshaking.

### *Simultaneous Open*

In a **simultaneous open,** both applications issue active opens. This is a rare situation in which there is no client or server; communication is between two peers that know their local port numbers. This case is allowed by TCP, but is unlikely to happen because both ends need to send SYN segments to each other and the segments are in transit simultaneously. This means that the two applications must issue active opens almost at the same time. Figure 15.18 shows the connection establishment phase for this scenario. Both TCPs go through **SYN-SENT** and **SYN-RCVD** states before going to the **ESTABLISHED** state. A close look shows that both processes act as client and server. The two SYN+ACK segments acknowledge the SYN segments and open the connection. Note that connection establishment involves a four-way handshake. The data transfer and the connection termination phases are the same as previous examples and are not shown in the figure. We leave the state transition diagram for this scenario as an exercise.

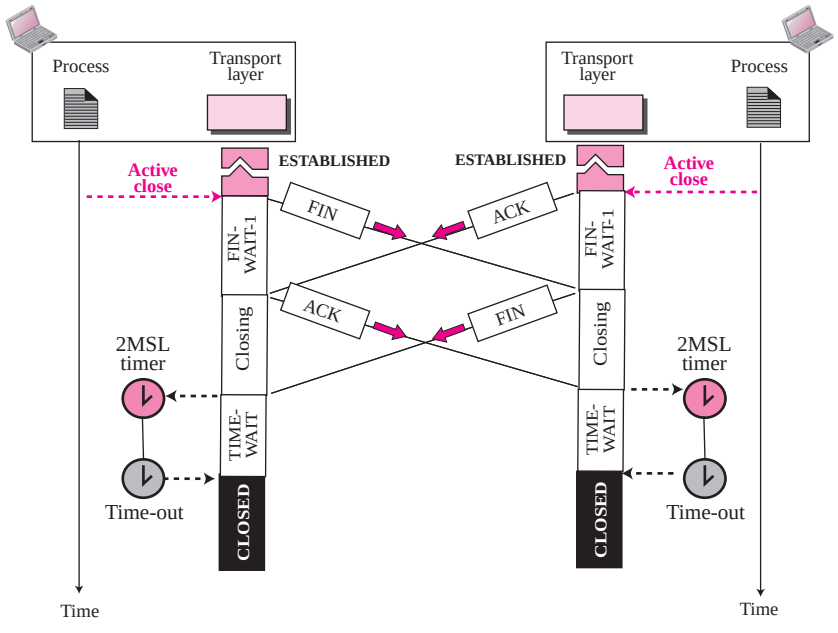
**Figure 15.18** *Simultaneous open*



### *Simultaneous Close*

Another uncommon, but possible, scenario is the **simultaneous close** shown in Figure 15.19.

**Figure 15.19** *Simultaneous close*

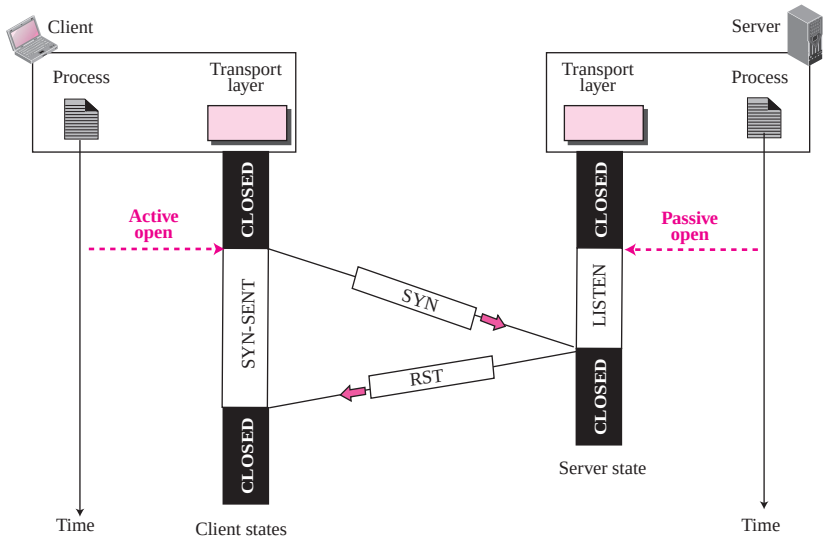



In this situation, both ends issue an active close. Both TCPs go to the **FIN-WAIT-1** state and send FIN segments that are in transit simultaneously. After receiving the FIN segment, each end goes to the **CLOSING** state and sends an ACK segment. The **CLOSING** state takes the place of **FIN-WAIT-2** or **CLOSE-WAIT** in a common scenario. After receiving the ACK segment, each end moves to the **TIME-WAIT** state. Note that this duration is required for both ends because each end has sent an ACK that may get lost. We have eliminated the connection establishment and the data transfer phases in the figure. We leave the state transition diagram in this case as an exercise.

### *Denying a Connection*

One common situation occurs when a server TCP denies the connection, perhaps because the destination port number in the SYN segment defines a server that is not in the **LISTEN** state at the moment. The server TCP, after receiving the SYN segment sends an RST+ACK segment that acknowledges the SYN segment, and, at the same time, resets (denies) the connection. It goes to the **LISTEN** state to wait for another connection. The client, after receiving the RST+ACK, goes to the **CLOSED** state. Figure 15.20 shows this situation.

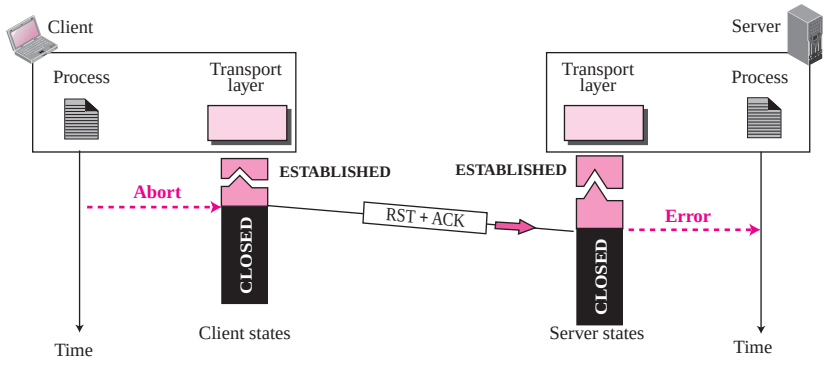
**Figure 15.20** *Denying a connection*




### *Aborting a Connection*

Figure 15.21 shows a situation in which the client process has issued an abort. Note that this feature is not shown in the general transition diagram of Figure 15.13 because it is something that can be optionally implemented by the vendor.

**Figure 15.21** *Aborting a connection*




A process can abort a connection instead of closing it. This can happen if the process has failed (perhaps locked up in an infinite loop) or does not want the data in the queue to be sent (due to some discrepancy in the data). TCP may also want to abort the connection. This can happen if it receives a segment belonging to a previous connection (incarnation). In all of these cases, the TCP can send an RST segment to abort the connection.

Its TCP sends an RST+ACK segment and throws away all data in the queue. The server TCP also throws away all queued data and informs the server process via an error message. Both TCPs go to the **CLOSED** state immediately. Note that no ACK segment is generated in response to the RST segment.

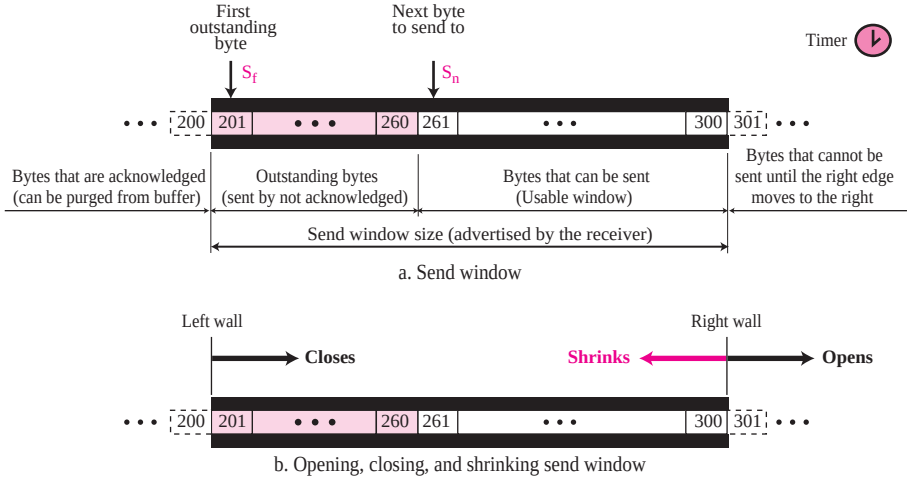
## 15.6 WINDOWS IN TCP

Before discussing data transfer in TCP and the issues such as flow, error, and congestion control, we describe the windows used in TCP. TCP uses two windows (send window and receive window) for each direction of data transfer, which means four windows for a bidirectional communication. However, to make the discussion simple, we make an unrealistic assumption that communication is only unidirectional (say from client to server); the bidirectional communication can be inferred using two unidirectional communications with piggybacking.

### Send Window

Figure 15.22 shows an example of a send window. The window we have used is of size 100 bytes (normally thousands of bytes), but later we see that the send window size is dictated by the receiver (flow control) and the congestion in the underlying network (congestion control). The figure shows how a send window *opens*, *closes*, or *shrinks*.

**Figure 15.22** *Send window in TCP*




The send window in TCP is similar to one used with the Selective Repeat protocol (Chapter 13), but with some differences:

1. One difference is the nature of entities related to the window. The window in SR numbers pockets, but the window in the TCP numbers bytes. Although actual

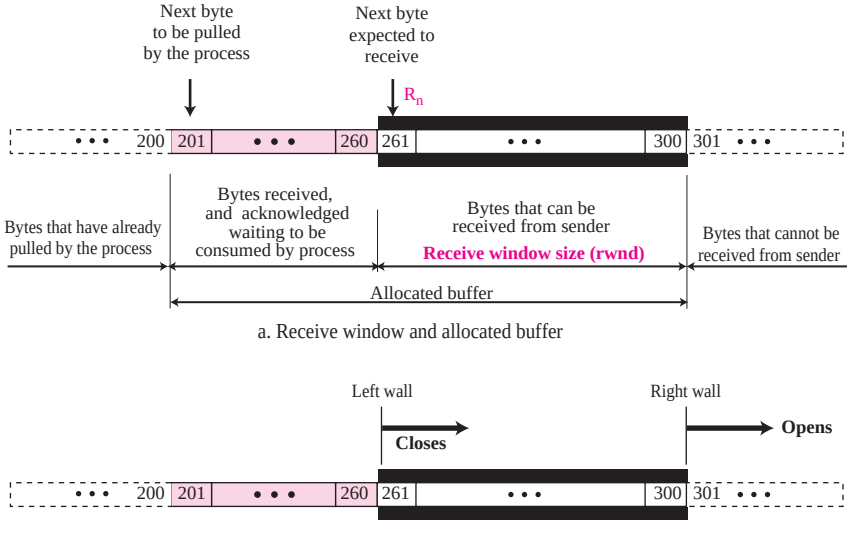
transmission in TCP occurs segment by segment, the variables that control the window are expressed in bytes.

2. The second difference is that, in some implementations, TCP can store data received from the process and send them later, but we assume that the sending TCP is capable of sending segments of data as soon as it receives them from its process.
3. Another difference is the number of timers. The theoretical Selective Repeat protocol may use several timers for each packet sent, but the TCP protocol uses only one timer. We later explain the use of this timer in error control.

## Receive Window

Figure 15.23 shows an example of a receive window. The window we have used is of size 100 bytes (normally thousands of bytes). The figure also shows how the receive window opens and closes; in practice, the window should never shrink.

**Figure 15.23** *Receive window in TCP*




There are two differences between the receive window in TCP and the one we used for SR in Chapter 13.

1. The first difference is that TCP allows the receiving process to pull data at its own pace. This means that part of the allocated buffer at the receiver may be occupied by bytes that have been received and acknowledged, but are waiting to be pulled by the receiving process. The receive window size is then always smaller or equal to the buffer size, as shown in the above figure. The receiver window size determines the number of bytes that the receive window can accept from the sender before being

overwhelmed (flow control). In other words, the receive window size, normally called *rwnd*, can be determined as:

**rwnd = buffer size – number of waiting bytes to be pulled**

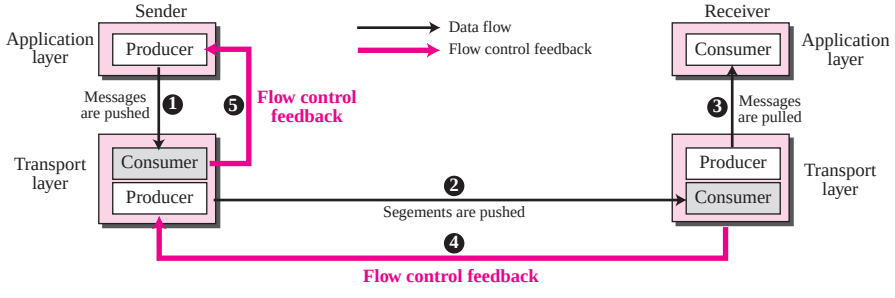
2. The second difference is the way acknowledgments are used in the TCP protocol. Remember that an acknowledgement in SR is selective, defining the uncorrupted packets that have been received. The major acknowledgment mechanism in TCP is a cumulative acknowledgment announcing the next expected byte to receive (in this way TCP looks like GBN discussed in Chapter 13). The new versions of TCP, however, uses both cumulative and selective acknowledgements as we will discuss later in the option section.

## 15.7 FLOW CONTROL

As discussed in Chapter 13, *flow control* balances the rate a producer creates data with the rate a consumer can use the data. TCP separates flow control from error control. In this section we discuss flow control, ignoring error control. We temporarily assume that the logical channel between the sending and receiving TCP is error-free.

Figure 15.24 shows unidirectional data transfer between a sender and a receiver; bidirectional data transfer can be deduced from unidirectional one as discussed in Chapter 13.

**Figure 15.24** *Data flow and flow control feedbacks in TCP*




The figure shows that data travel from the sending process down to the sending TCP, from the sending TCP to the receiving TCP, and from receiving TCP up to the receiving process (paths 1, 2, and 3). Flow control feedbacks, however, are traveling from the receiving TCP to the sending TCP and from the sending TCP up to the sending process (paths 4 and 5). Most implementations of TCP do not provide flow control feedback from the receiving process to the receiving TCP; they let the receiving process pull data from the receiving TCP whenever it is ready to do so. In other words, the receiving TCP controls the sending TCP; the sending TCP controls the sending process.

Flow control feedback from the sending TCP to the sending process (path 5) is achieved through simple rejection of data by sending TCP when its window is full. This means that our discussion of flow control concentrates on the feedback sent from the receiving TCP to the sending TCP (path 4).

## Opening and Closing Windows

To achieve flow control, TCP forces the sender and the receiver to adjust their window sizes, although the size of the buffer for both parties is fixed when the connection is established. The receive window closes (moves its left wall to the right) when more bytes arrive from the sender; it opens (moves its right wal l to the right) when more bytes are pulled by the process. We assume that it does not shrink (the right wall does not move to the left).

The opening, closing, and shrinking of the send window is controlled by the receiver. The send window closes (moves its left wall to the right) when a new acknowledgement allows it to do so. The send window opens (its right wall moves to the right) when the receive window size (rwnd) advertised by the receiver allows it to do so. The send window shrinks on occasion. We assume that this situation does not occur.

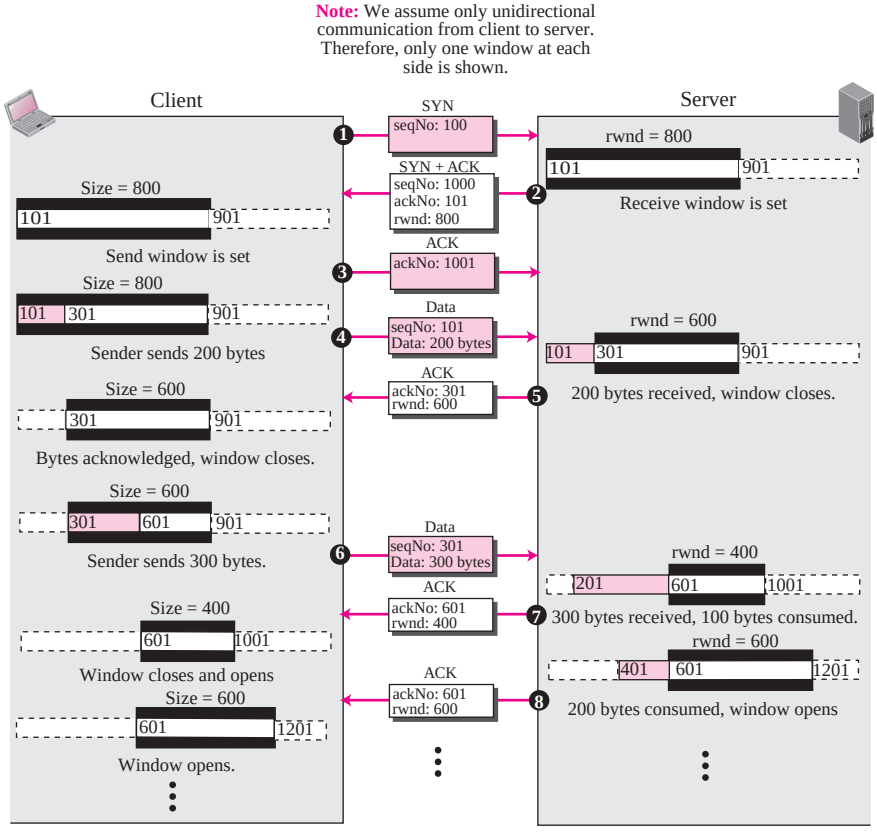
### *A Scenario*

We show how the send and receive windows are set during the connection establishment phase, and how their situations will change during data transfer. Figure 15.25 shows a simple example of unidirectional data transfer (from client to server). For the time being, we ignore error control, assuming that no segment is corrupted, lost, duplicated, or arrived out of order. Note that we have shown only two windows for unidirectional data transfer.

Eight segments are exchanged between the client and server:

1. The first segment is from the client to the server (a SYN segment) to request connection. The client announces its initial seqNo = 100. When this segment arrives at the server, it allocates a buffer size of 800 (an assumption) and sets its window to cover the whole buffer (rwnd = 800). Note that the number of the next byte to arrive is 101.
2. The second segment is from the server to the client. This is an ACK + SYN segment. The segment uses ackNo = 101 to show that it expects to receive bytes starting from 101. It also announces that the client can set a buffer size of 800 bytes.
3. The third segment is the ACK segment from the client to the server.
4. After the client has set its window with the size (800) dictated by the server, the process pushes 200 bytes of data. The TCP client numbers these bytes 101 to 300. It then creates a segment and sends it to the server. The segment shows the starting byte number as 101 and the segment carries 200 bytes. The window of the client is then adjusted to show 200 bytes of data are sent but waiting for acknowledgment. When this segment is received at the server, the bytes are stored, and the receive window closes to show that the next byte expected is byte 301; the stored bytes occupy 200 bytes of buffer.

**Figure 15.25** *An example of flow control*




5. The fifth segment is the feedback from the server to the client. The server acknowledges bytes up to and including 300 (expecting to receive byte 301). The segment also carries the size of the receive window after decrease (600). The client, after receiving this segment, purges the acknowledged bytes from its window and closes its window to show that the next byte to send is byte 301. The window size, however, decreases to 600 bytes. Although the allocated buffer can store 800 bytes, the window cannot open (moving its right wall to the right) because the receiver does not let it.
6. Segment 6 is sent by the client after its process pushes 300 more bytes. The segment defines seqNo as 301 and contains 300 bytes. When this segment arrives at the server, the server stores them, but it has to reduce its window size. After its process has pulled 100 bytes of data, the window closes from the left for the amount of 300 bytes, but opens from the right for the amount of 100 bytes. The result is that the size is only reduced 200 bytes. The receiver window size is now 400 bytes.
7. In segment 7, the server acknowledges the receipt of data, and announces that its window size is 400. When this segment arrives at the client, the client has no choice but to reduce its window again and set the window size to the value of rwnd = 400

advertised by the server. The send window closes from the left by 300 bytes, and opens from the right by 100 bytes.

8. Segment 8 is also from the server after its process has pulled another 200 bytes. Its window size increases. The new rwnd value is now 600. The segment informs the client that the server still expects byte 601, but the server window size has expanded to 600. We need to mention that the sending of this segment depends on the policy imposed by the implementation. Some implementations may not allow advertisement of the rwnd at this time; the server then needs to receive some data before doing so. After this segment arrives at the client, the client opens its window by 200 bytes without closing it. The result is that its window size increases to 600 bytes.

## Shrinking of Windows

As we said before, the receive window cannot shrink. But the send window can shrink if the receiver defines a value for rwnd that results in shrinking the window. Some implementations do not allow the shrinking of the send window. The limitation does not allow the right wall of the send window to move to the left. In other words, the receiver needs to keep the following relationship between the last and new acknowledgment and the last and new rwnd values to prevent the shrinking of the send window:

$$\textbf{new ackNo} + \textbf{new rwnd} \geq \textbf{last ackNo} + \textbf{last rwnd}$$

The left side of the inequality represents the new position of the right wall with respect to the sequence number space; the right side shows the old position of the right wall. The relationship shows that the right wall should not move to the left. The inequality is a mandate for the receiver to check its advertisment. However, note that the inequality is valid only if $S_f < S_n$; we need to remember that all calculations are in modulo $2^{32}$.

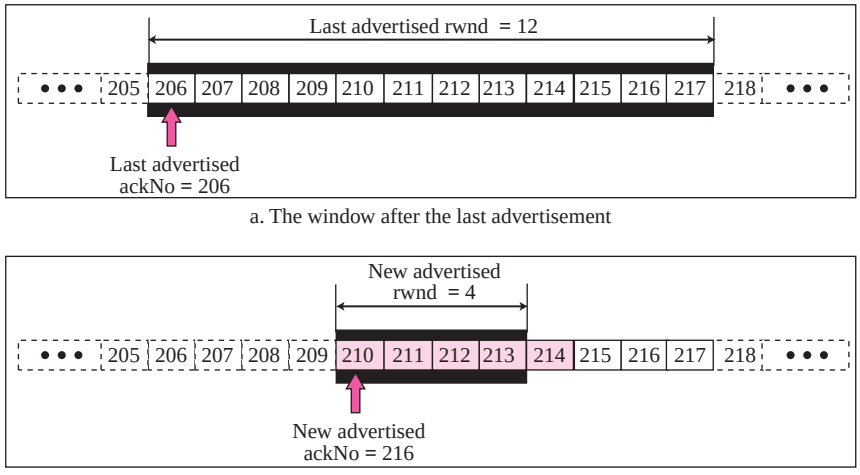
### Example 15.2

Figure 15.26 shows the reason for this mandate by some implementations. Part a of the figure shows values of last acknowledgment and rwnd. Part b shows the situation in which the sender has sent bytes 206 to 214. Bytes 206 to 209 are acknowledged and purged. The new advertisment, however, defines the new value of rwnd as 4, in which $210 + 4 < 206 + 12$. When the send window shrinks, it creates a problem: byte 214 which has been already sent is outside the window. The relation discussed before forces the receiver to maintain the right-hand wall of the window to be as shown in part a because the receiver does not know which of the bytes 210 to 217 has already been sent. One way to prevent this situation is to let the receiver postpone its feedback until enough buffer locations are available in its window. In other words, the receiver should wait until more bytes are consumed by its process to meet the relationship described above.

### *Window Shutdown*

We said that shrinking the send window by moving its right wall to the left is strongly discouraged. However, there is one exception: the receiver can temporarily shut down the window by sending a rwnd of 0. This can happen if for some reason the receiver does not want to receive any data from the sender for a while. In this case, the sender does not actually shrink the size of the window, but stops sending data until a new advertisement has arrived. As we will see later, even when the window is shut down by

**Figure 15.26** *Example 15.2*




an order from the receiver, the sender can always send a segment with 1 byte of data. This is called probing and is used to prevent a deadlock (see the section on TCP timers).

## Silly Window Syndrome

A serious problem can arise in the sliding window operation when either the sending application program creates data slowly or the receiving application program consumes data slowly, or both. Any of these situations results in the sending of data in very small segments, which reduces the efficiency of the operation. For example, if TCP sends segments containing only 1 byte of data, it means that a 41-byte datagram (20 bytes of TCP header and 20 bytes of IP header) transfers only 1 byte of user data. Here the overhead is 41/1, which indicates that we are using the capacity of the network very inefficiently. The inefficiency is even worse after accounting for the data link layer and physical layer overhead. This problem is called the **silly window syndrome.** For each site, we first describe how the problem is created and then give a proposed solution.

### *Syndrome Created by the Sender*

The sending TCP may create a silly window syndrome if it is serving an application program that creates data slowly, for example, 1 byte at a time. The application program writes 1 byte at a time into the buffer of the sending TCP. If the sending TCP does not have any specific instructions, it may create segments containing 1 byte of data. The result is a lot of 41-byte segments that are traveling through an internet.

The solution is to prevent the sending TCP from sending the data byte by byte. The sending TCP must be forced to wait and collect data to send in a larger block. How long should the sending TCP wait? If it waits too long, it may delay the process. If it does not wait long enough, it may end up sending small segments. Nagle found an elegant solution.

**Nagle's Algorithm** **Nagle's algorithm** is simple:

1. The sending TCP sends the first piece of data it receives from the sending application program even if it is only 1 byte.
2. After sending the first segment, the sending TCP accumulates data in the output buffer and waits until either the receiving TCP sends an acknowledgment or until enough data has accumulated to fill a maximum-size segment. At this time, the sending TCP can send the segment.
3. Step 2 is repeated for the rest of the transmission. Segment 3 is sent immediately if an acknowledgment is received for segment 2, or if enough data have accumulated to fill a maximum-size segment.

The elegance of Nagle's algorithm is in its simplicity and in the fact that it takes into account the speed of the application program that creates the data and the speed of the network that transports the data. If the application program is faster than the network, the segments are larger (maximum-size segments). If the application program is slower than the network, the segments are smaller (less than the maximum segment size).

### *Syndrome Created by the Receiver*

The receiving TCP may create a silly window syndrome if it is serving an application program that consumes data slowly, for example, 1 byte at a time. Suppose that the sending application program creates data in blocks of 1 kilobyte, but the receiving application program consumes data 1 byte at a time. Also suppose that the input buffer of the receiving TCP is 4 kilobytes. The sender sends the first 4 kilobytes of data. The receiver stores it in its buffer. Now its buffer is full. It advertises a window size of zero, which means the sender should stop sending data. The receiving application reads the first byte of data from the input buffer of the receiving TCP. Now there is 1 byte of space in the incoming buffer. The receiving TCP announces a window size of 1 byte, which means that the sending TCP, which is eagerly waiting to send data, takes this advertisement as good news and sends a segment carrying only 1 byte of data. The procedure will continue. One byte of data is consumed and a segment carrying 1 byte of data is sent. Again we have an efficiency problem and the silly window syndrome.

Two solutions have been proposed to prevent the silly window syndrome created by an application program that consumes data slower than they arrive.

**Clark's Solution** **Clark's solution** is to send an acknowledgment as soon as the data arrive, but to announce a window size of zero until either there is enough space to accommodate a segment of maximum size or until at least half of the receive buffer is empty.

**Delayed Acknowledgment** The second solution is to delay sending the acknowledgment. This means that when a segment arrives, it is not acknowledged immediately. The receiver waits until there is a decent amount of space in its incoming buffer before acknowledging the arrived segments. The delayed acknowledgment prevents the sending TCP from sliding its window. After the sending TCP has sent the data in the window, it stops. This kills the syndrome.

Delayed acknowledgment also has another advantage: it reduces traffic. The receiver does not have to acknowledge each segment. However, there also is a disadvantage in

that the delayed acknowledgment may result in the sender unnecessarily retransmitting the unacknowledged segments.

TCP balances the advantages and disadvantages. It now defines that the acknowledgment should not be delayed by more than 500 ms.

## 15.8 ERROR CONTROL

TCP is a reliable transport layer protocol. This means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated.

TCP provides reliability using error control. Error control includes mechanisms for detecting and resending corrupted segments, resending lost segments, storing out-of-order segments until missing segments arrive, and detecting and discarding duplicated segments. Error control in TCP is achieved through the use of three simple tools: checksum, acknowledgment, and time-out.

### Checksum

Each segment includes a checksum field, which is used to check for a corrupted segment. If a segment is corrupted as deleted by an invalid checksum, the segment is discarded by the destination TCP and is considered as lost. TCP uses a 16-bit checksum that is mandatory in every segment. We discussed how to calculate checksums earlier in the chapter.

### Acknowledgment

TCP uses acknowledgments to confirm the receipt of data segments. Control segments that carry no data, but consume a sequence number, are also acknowledged. ACK segments are never acknowledged.

**ACK segments do not consume sequence numbers and are not acknowledged.**

#### *Acknowledgment Type*

In the past, TCP used only one type of acknowledgment: cumulative acknowledgment. Today, some TCP implementations also use selective acknowledgment.

**Cumulative Acknowledgment (ACK)** TCP was originally designed to acknowledge receipt of segments cumulatively. The receiver advertises the next byte it expects to receive, ignoring all segments received and stored out of order. This is sometimes referred to as positive cumulative acknowledgment or ACK. The word "positive" indicates that no feedback is provided for discarded, lost, or duplicate segments. The 32-bit ACK field in the TCP header is used for cumulative acknowledgments and its value is valid only when the ACK flag bit is set to 1.

**Selective Acknowledgment (SACK)** More and more implementations are adding another type of acknowledgment called **selective acknowledgment** or **SACK.** A

SACK does not replace ACK, but reports additional information to the sender. A SACK reports a block of data that is out of order, and also a block of segments that is duplicated, i.e. received more than once. However, since there is no provision in the TCP header for adding this type of information, SACK is implemented as an option at the end of the TCP header. We discuss this new feature when we discuss options in TCP.

### *Generating Acknowledgments*

When does a receiver generate acknowledgments? During the evolution of TCP, several rules have been defined and used by several implementations. We give the most common rules here. The order of a rule does not necessarily define its importance.

1. When end A sends a data segment to end B, it must include (piggyback) an acknowledgment that gives the next sequence number it expects to receive. This rule decreases the number of segments needed and therefore reduces traffic.
2. When the receiver has no data to send and it receives an in-order segment (with expected sequence number) and the previous segment has already been acknowledged, the receiver delays sending an ACK segment until another segment arrives or until a period of time (normally 500 ms) has passed. In other words, the receiver needs to delay sending an ACK segment if there is only one outstanding in-order segment. This rule reduces ACK segment traffic.
3. When a segment arrives with a sequence number that is expected by the receiver, and the previous in-order segment has not been acknowledged, the receiver immediately sends an ACK segment. In other words, there should not be more than two in-order unacknowledged segments at any time. This prevents the unnecessary retransmission of segments that may create congestion in the network.
4. When a segment arrives with an out-of-order sequence number that is higher than expected, the receiver immediately sends an ACK segment announcing the sequence number of the next expected segment. This leads to the *fast retransmission* of missing segments (discussed later).
5. When a missing segment arrives, the receiver sends an ACK segment to announce the next sequence number expected. This informs the receiver that segments reported missing have been received.
6. If a duplicate segment arrives, the receiver discards the segment, but immediately sends an acknowledgment indicating the next in-order segment expected. This solves some problems when an ACK segment itself is lost.

## Retransmission

The heart of the error control mechanism is the retransmission of segments. When a segment is sent, it is stored in a queue until it is acknowledged. When the retransmission timer expires or when the sender receives three duplicate ACKs for the first segment in the queue, that segment is retransmitted.

### *Retransmission after RTO*

The sending TCP maintains one **retransmission time-out (RTO)** for each connection. When the timer matures, i.e. times out, TCP sends the segment in the front of the queue (the segment with the smallest sequence number) and restarts the timer. Note that again

we assume $S_f < S_n$. This version of TCP is sometimes referred to as *Tahoe*. We will see later that the value of RTO is dynamic in TCP and is updated based on the round-trip time (RTT) of segments. RTT is the time needed for a segment to reach a destination and for an acknowledgment to be received.

### *Retransmission after Three Duplicate ACK Segments*

The previous rule about retransmission of a segment is sufficient if the value of RTO is not large. To help throughput by allowing sender to retransmit sooner than waiting for a time out, most implementations today follow the three duplicate ACKs rule and retransmit the missing segment immediately. This feature is called **fast retransmission,** and the version of TCP that uses this feature is referred to as Reno. In this version, if three duplicate acknowledgments (i.e., an original ACK plus three exactly identical copies) arrives for a segment, the next segment is retransmitted without waiting for the time-out.

## Out-of-Order Segments

TCP implementations today do not discard out-of-order segments. They store them temporarily and flag them as out-of-order segments until the missing segments arrive. Note, however, that out-of-order segments are never delivered to the process. TCP guarantees that data are delivered to the process in order.

**Data may arrive out of order and be temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order data are delivered to the process.**

## FSMs for Data Transfer in TCP

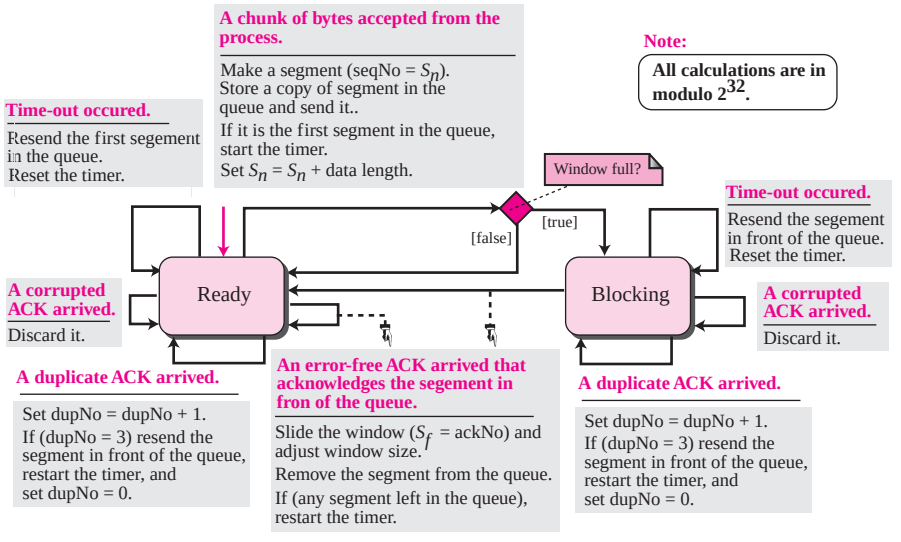
Data transfer in TCP is close to the Selective Repeat protocol (discussed in Chapter 13) with a slight similarity to GBN. Since TCP accepts out-of-order segments, TCP can be thought of as behaving like the SR protocol, but since the original acknowledgments are cumulative, it looks like GBN. However, if the TCP implementation uses SACKs, as discussed later, then TCP is closest to SR.

**TCP can be best modeled as a Selective Repeat protocol.**

### *Sender-Side FSM*

Let us show a simplified FSM for the sender side of the TCP protocol similar to the one we discussed for SR protocol with some changes specific to TCP. We assume that the communication is unidirectional and the segments are acknowledged using ACK segments. We also ignore selective acknowledgments and congestion control for the moment. Figure 15.27 shows the simplified FSM for the sender site. Note that the FSM is rudimentary; it does not include issues such as silly window syndrome (Nagle's algorithm) or window shutdown (discussed later). It defines a unidirectional communication, ignoring all issues that affect bidirectional communication.

**Figure 15.27** *Simplified FSM for the TCP sender side*




There are some differences between the above FSM and the one we discussed for an SR protocol in Chapter 13. One difference is the fast transmission (three duplicate ACKs). The other is the window size adjustment based on the value of rwnd (ignoring congestion control for the moment).

### *Receiver-Side FSM*

Now let us show a simplified FSM for the receiver-side TCP protocol similar to the one we discuss for SR protocol with some changes specific to TCP. We assume that the communication is unidirectional and the segments are acknowledged using ACK segments. We also ignore the selective acknowledgment and congestion control for the moment. Figure 15.28 shows the simplified FSM for the receiver. Note that we ignore some issues such as silly window syndrome (Clark's solution) and window shutdown.

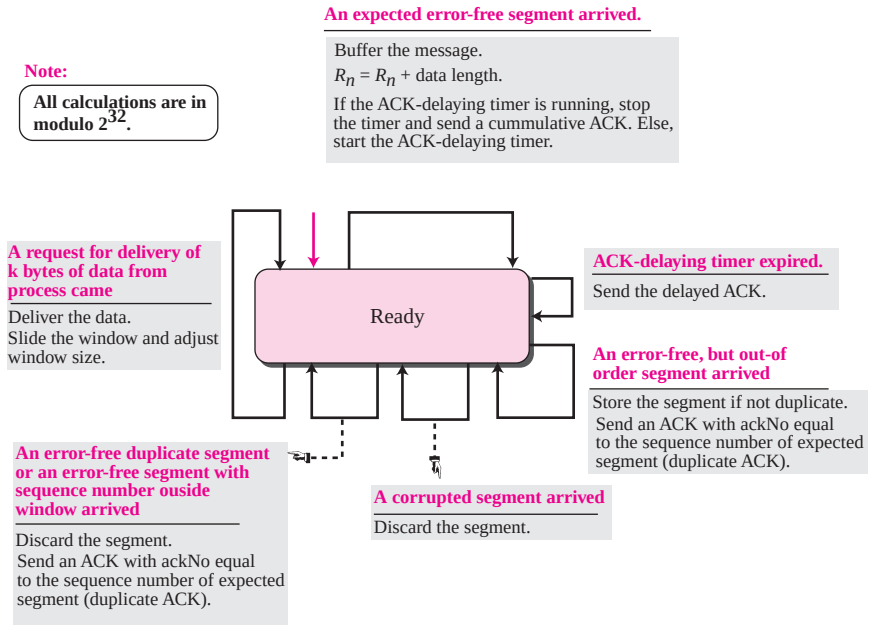
Again, there are some differences between this FSM and the one we discussed for an SR protocol in Chapter 13. One difference is the ACK delaying in unidirectional communication. The other difference is the sending of duplicate ACKs to allow the sender to implement fast transmission policy.

We also need to emphasize that bidirectional FSM for the receiver is not as simple as the one for SR; we need to consider some policies such as sending an immediate ACK if the receiver has some data to return.

## Some Scenarios

In this section we give some examples of scenarios that occur during the operation of TCP, considering only error control issues. In these scenarios, we show a segment by a rectangle. If the segment carries data, we show the range of byte numbers and the value

**Figure 15.28** *Simplified FSM for the TCP receiver side*




of the acknowledgment field. If it carries only an acknowledgment, we show only the acknowledgment number in a smaller box.

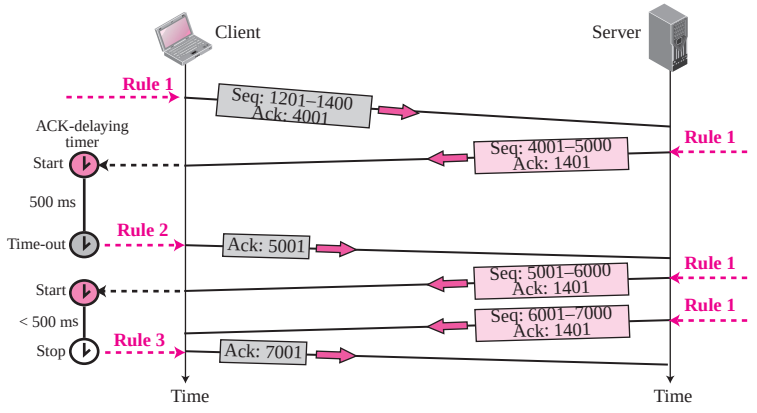
### *Normal Operation*

The first scenario shows bidirectional data transfer between two systems as shown in Figure 15.29. The client TCP sends one segment; the server TCP sends three. The figure shows which rule applies to each acknowledgment. For the client's first segment and all three server segments, rule 1 applies. There are data to be sent so the segment displays the next byte expected. When the client receives the first segment from the server, it does not have any more data to send; it needs to send only an ACK segment. However, according to rule 2, the acknowledgment needs to be delayed for 500 ms to see if any more segments arrive. When the ACK-delaying timer matures, it triggers an acknowledgment. This is because the client has no knowledge if other segments are coming; it cannot delay the acknowledgment forever. When the next segment arrives another ACK-delaying timer is set. However, before it matures, the third segment arrives. The arrival of the third segment triggers another acknowledgment based on rule 3. We have not shown the RTO timer because no segment is lost or delayed. We just assume that this timer performs its duty.

### *Lost Segment*

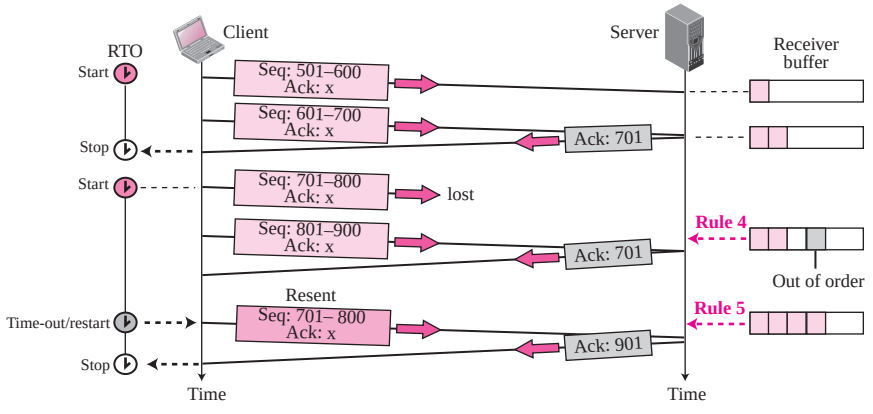
In this scenario, we show what happens when a segment is lost or corrupted. A lost or corrupted segment is treated the same way by the receiver. A lost segment is discarded somewhere in the network; a corrupted segment is discarded by the receiver itself. Both

**Figure 15.29** *Normal operation*




are considered lost. Figure 15.30 shows a situation in which a segment is lost (probably discarded by some router in the network due to congestion).

**Figure 15.30** *Lost segment*




We are assuming that data transfer is unidirectional: one site is sending, the other receiving. In our scenario, the sender sends segments 1 and 2, which are acknowledged immediately by an ACK (rule 3). Segment 3, however, is lost. The receiver receives segment 4, which is out of order. The receiver stores the data in the segment in its buffer but leaves a gap to indicate that there is no continuity in the data. The receiver immediately sends an acknowledgment to the sender displaying the next byte it expects (rule 4). Note that the receiver stores bytes 801 to 900, but never delivers these bytes to the application until the gap is filled.

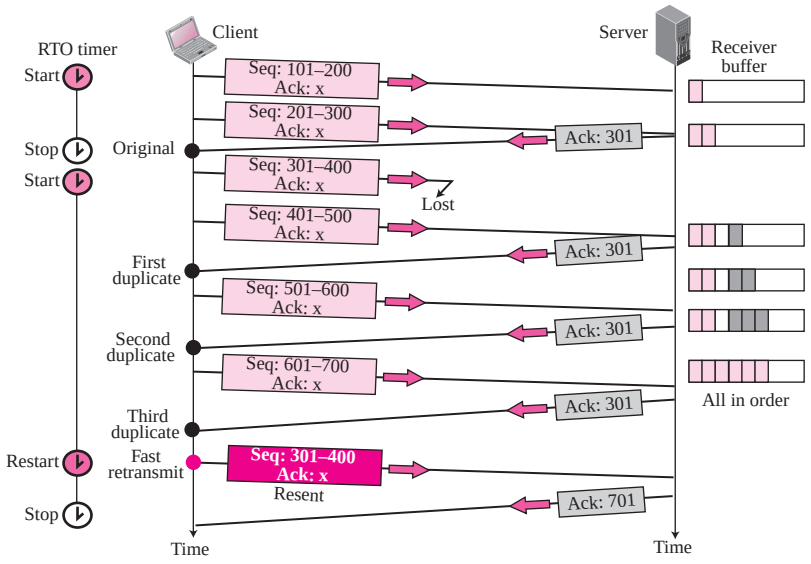
**The receiver TCP delivers only ordered data to the process.**

The sender TCP keeps one RTO timer for the whole period of connection. When the third segment times out, the sending TCP resends segment 3, which arrives this time and is acknowledged properly (rule 5).

### *Fast Retransmission*

In this scenario, we want to show *fast retransmission.* Our scenario is the same as the second except that the RTO has a larger value (see Figure 15.31).

**Figure 15.31** *Fast retransmission*




Each time the receiver receives the fourth, fifth, and sixth segments, it triggers an acknowledgment (rule 4). The sender receives four acknowledgments with the same value (three duplicates). Although the timer has not matured, the rule for fast transmission requires that segment 3, the segment that is expected by all of these duplicate acknowledgments, be resent immediately. After resending this segment, the timer is restarted.

### *Delayed Segment*

The fourth scenario features a delayed segment. TCP uses the services of IP, which is a connectionless protocol. Each IP datagram encapsulating a TCP segment may reach the final destination through a different route with a different delay. Hence TCP segments may be delayed. Delayed segments sometimes may time out. If the delayed segment arrives after it has been resent, it is considered a duplicate segment and discarded.

### *Duplicate Segment*

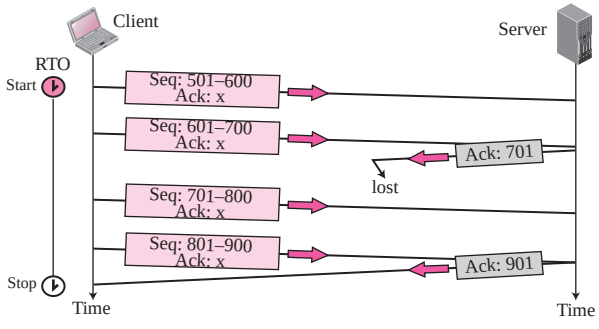
A duplicate segment can be created, for example, by a sending TCP when a segment is delayed and treated as lost by the receiver. Handling the duplicated segment is a simple process for the destination TCP. The destination TCP expects a continuous stream of

bytes. When a segment arrives that contains a sequence number equal to an already received and stored segment, it is discarded. An ACK is sent with ackNo defining the expected segment.

### *Automatically Corrected Lost ACK*

This scenario shows a situation in which information in lost acknowledgment is contained in the next one, a key advantage of using cumulative acknowledgments. Figure 15.32 shows a lost acknowledgment sent by the receiver of data. In the TCP acknowledgment mechanism, a lost acknowledgment may not even be noticed by the source TCP. TCP uses an accumulative acknowledgment system. We can say that the next acknowledgment automatically corrects the loss of the acknowledgment.

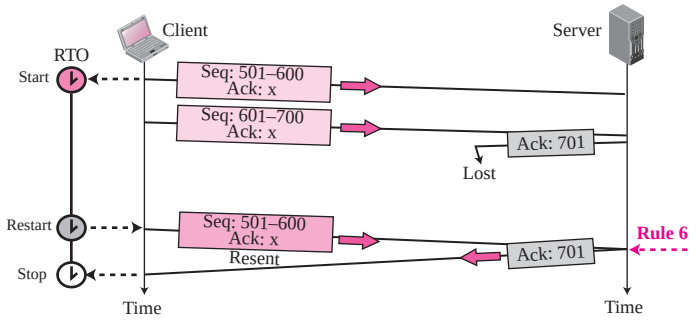
**Figure 15.32** *Lost acknowledgment*




### *Lost Acknowledgment Corrected by Resending a Segment*

Figure 15.33 shows a scenario in which an acknowledgment is lost.

**Figure 15.33** *Lost acknowledgment corrected by resending a segment*




If the next acknowledgment is delayed for a long time or there is no next acknowledgment (the lost acknowledgment is the last one sent), the correction is triggered by the RTO timer. A duplicate segment is the result. When the receiver receives a duplicate

segment, it discards it, and resends the last ACK immediately to inform the sender that the segment or segments have been received.

Note that only one segment is retransmitted although two segments are not acknowledged. When the sender receives the retransmitted ACK, it knows that both segments are safe and sound because acknowledgment is cumulative.

#### *Deadlock Created by Lost Acknowledgment*

There is one situation in which loss of an acknowledgment may result in system deadlock. This is the case in which a receiver sends an acknowledgment with *rwnd* set to 0 and requests that the sender shut down its window temporarily. After a while, the receiver wants to remove the restriction; however, if it has no data to send, it sends an ACK segment and removes the restriction with a nonzero value for *rwnd*. A problem arises if this acknowledgment is lost. The sender is waiting for an acknowledgment that announces the nonzero *rwnd*. The receiver thinks that the sender has received this and is waiting for data. This situation is called a **deadlock;** each end is waiting for a response from the other end and nothing is happening. A retransmission timer is not set. To prevent deadlock, a persistence timer was designed that we will study later in the chapter.

**Lost acknowledgments may create deadlock if they are not properly handled.**

## 15.9 CONGESTION CONTROL

We briefly discussed congestion control in Chapter 13. Congestion control in TCP is based on both open-loop and closed-loop mechanisms. TCP uses a congestion window and a congestion policy that avoid congestion and detect and alleviate congestion after it has occurred.

### Congestion Window

Previously, we talked about flow control and tried to discuss solutions when the receiver is overwhelmed with data. We said that the sender window size is determined by the available buffer space in the receiver (rwnd). In other words, we assumed that it is only the receiver that can dictate to the sender the size of the sender's window. We totally ignored another entity here, the network. If the network cannot deliver the data as fast as it is created by the sender, it must tell the sender to slow down. In other words, in addition to the receiver, the network is a second entity that determines the size of the sender's window.

The sender has two pieces of information: the receiver-advertised window size and the congestion window size. The actual size of the window is the minimum of these two.

**Actual window size = minimum (rwnd, cwnd)**

We show shortly how the size of the congestion window (cwnd) is determined.

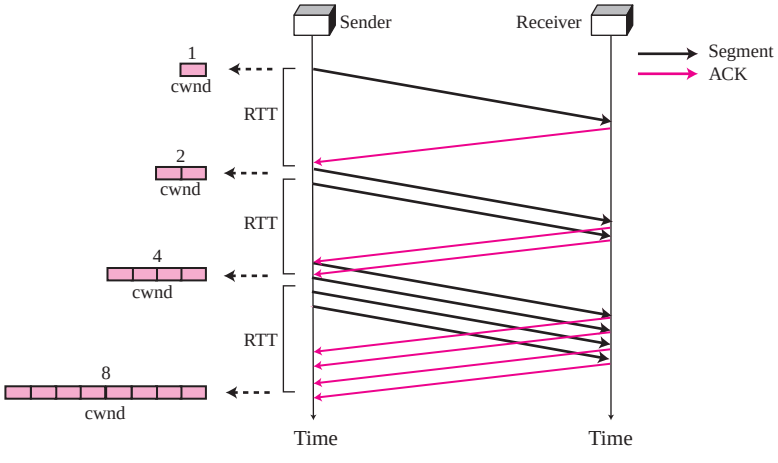
## Congestion Policy

TCP's general policy for handling congestion is based on three phases: slow start, congestion avoidance, and congestion detection. In the slow start phase, the sender starts with a slow rate of transmission, but increases the rate rapidly to reach a threshold. When the threshold is reached, the rate of increase is reduced. Finally if ever congestion is detected, the sender goes back to the slow start or congestion avoidance phase, based on how the congestion is detected.

### *Slow Start: Exponential Increase*

The **slow start** algorithm is based on the idea that the size of the congestion window (cwnd) starts with one maximum segment size (MSS). The MSS is determined during connection establishment using an option of the same name. The size of the window increases one MSS each time one acknowledgement arrives. As the name implies, the algorithm starts slowly, but grows exponentially. To show the idea let us look at Figure 15.34. We assume that *rwnd* is much longer than *cwnd*, so that the sender window size always equals *cwnd*. For simplicity, we ignore delayed-ACK policy and assume that each segment is acknowledged individually.

**Figure 15.34** *Slow start, exponential increase*




The sender starts with cwnd = 1 MSS. This means that the sender can send only one segment. After the first ACK arrives, the size of the congestion window is increased by 1, which means that cwnd is now 2. Now two more segments can be sent. When two more ACKs arrive, the size of the window is increased by 1 MSS for each ACK, which means cwnd is now 4. Now four more segments can be sent. When four ACKs arrive, the size of the window increases by 4, which means that cwnd is now 8.

If we look at the size of the cwnd in terms of round-trip times (RTTs), we find that the growth rate is exponential as shown below:

| | | |
|---|---|---|
| **Start** | → | **cwnd = 1** |
| **After 1 RTT** | → | **cwnd = 1 × 2 = 2 → $2^1$** |
| **After 2 RTT** | → | **cwnd = 2 × 2 = 4 → $2^2$** |
| **After 3 RTT** | → | **cwnd = 4 × 2 = 8 → $2^3$** |

We need, however, to mention that the slow start strategy is slower in the case of delayed acknowledgments. Remember, for each ACK, the cwnd is increased by only 1 MSS. Hence, if three segments are acknowledged accummlatively, the size of the cwnd increases by only 1 MSS, not 3 MSS. The growth is still exponential, but it is not a power of 2. With one ACK for every 2 segments, the power is closer to 1.5.

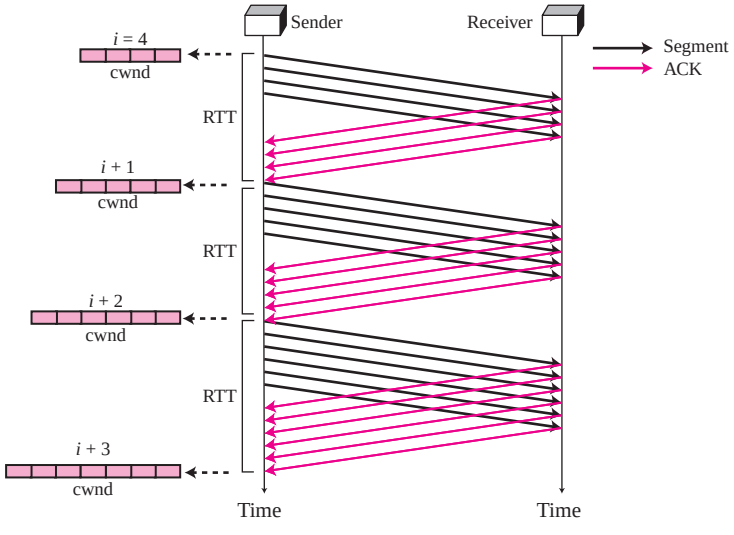
Slow start cannot continue indefinitely. There must be a threshold to stop this phase. The sender keeps track of a variable named *ssthresh* (slow start threshold). When the size of window in bytes reaches this threshold, slow start stops and the next phase starts.

**In the slow start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.**

### *Congestion Avoidance: Additive Increase*

If we start with the slow start algorithm, the size of the congestion window increases exponentially. To avoid congestion before it happens, one must slow down this exponential growth. TCP defines another algorithm called **congestion avoidance,** which increases the cwnd additively instead of exponentially. Figure 15.35 shows the idea.

**Figure 15.35** *Congestion avoidance, additive increase*



When the size of the congestion window reaches the slow start threshold in the case where cwnd = *i*, the slow start phase stops and the additive phase begins. In this algorithm, each time the whole "window" of segments is acknowledged, the size of the congestion window is increased by one. A window is the number of segments transmitted during RTT. In other words, the increase is based on RTT, not on the number of arrived ACKs. To show the idea, we apply this algorithm to the same scenario as slow start. In this case, after the sender has received acknowledgments for a complete window-size of segments, the size of the window is increased one segment. If we look at the size of cwnd in terms of round-trip time (RTT), we find that the rate is additive as shown below:

| | | |
|---|---|---|
| **Start** | → | **cwnd = *i*** |
| **After 1 RTT** | → | **cwnd = *i* + 1** |
| **After 2 RTT** | → | **cwnd = *i* + 2** |
| **After 3 RTT** | → | **cwnd = *i* + 3** |

**In the congestion avoidance algorithm the size of the congestion window increases additively until congestion is detected.**

### *Congestion Detection: Multiplicative Decrease*

If congestion occurs, the congestion window size must be decreased. The only way a sender can guess that congestion has occurred is the need to retransmit a segment. This is a major assumption made by TCP. Retransmission is needed to recover a missing packet which is assumed to have been dropped (i.e., lost) by a router that had so many incoming packets, that had to drop the missing segment, i.e., the router/network became overloaded or congested. However, retransmission can occur in one of two cases: when the RTO timer times out or when three duplicate ACKs are received. In both cases, the size of the threshold is dropped to half (**multiplicative decrease**). Most TCP implementations have two reactions:

1. If a time-out occurs, there is a stronger possibility of congestion; a segment has probably been dropped in the network and there is no news about the following sent segments. In this case TCP reacts strongly:
   - **a.** It sets the value of the threshold to half of the current window size.
   - **b.** It reduces cwnd back to one segment.
   - **c.** It starts the slow start phase again.
2. If three duplicate ACKs are received, there is a weaker possibility of congestion; a segment may have been dropped but some segments after that have arrived safely since three duplicate ACKs are received. This is called fast transmission and fast recovery. In this case, TCP has a weaker reaction as shown below:
   - **a.** It sets the value of the threshold to half of the current window size.
   - **b.** It sets cwnd to the value of the threshold (some implementations add three segment sizes to the threshold).
   - **c.** It starts the congestion avoidance phase.

### *Summary*

In Figure 15.36, we summarize the congestion policy of TCP and the relationships between the three phases. We give an example in Figure 15.37. We assume that the

**Figure 15.36** *TCP congestion policy summary*

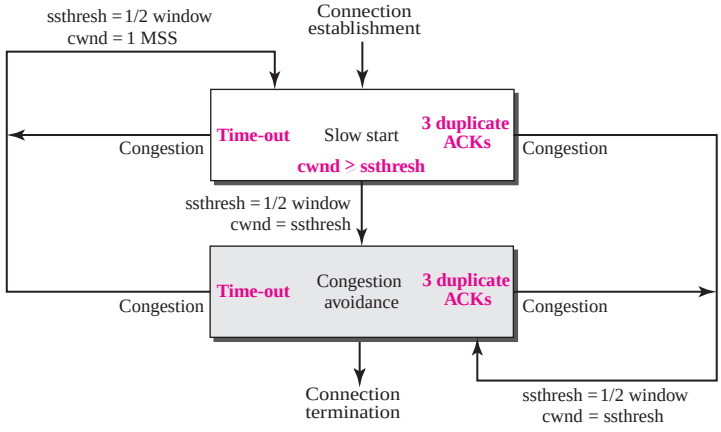
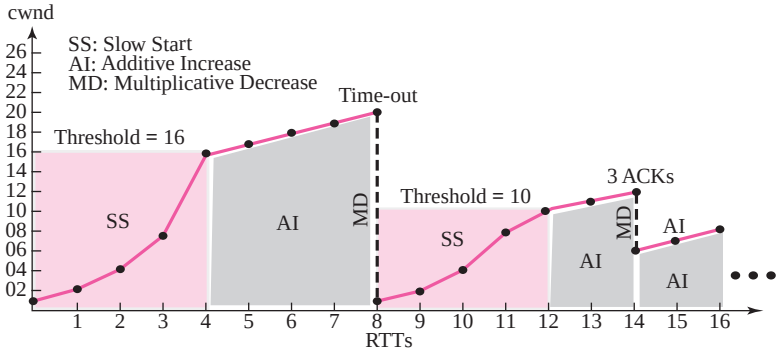



**Figure 15.37** *Congestion example*



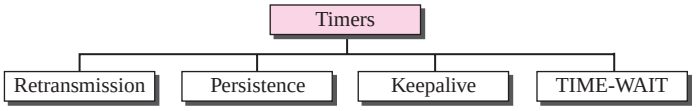

maximum window size is initially 32 segments. The threshold is initially set to 16 segments (half of the maximum window size). In the *slow start* phase the window size starts from 1 and grows exponentially until it reaches the threshold. After reaching the threshold, the *congestion avoidance (additive increase)* procedure allows the window size to increase linearly until a time-out occurs or the maximum window size is reached. In the figure, a time-out occurs when the window size is 20. At this moment, the *multiplicative decrease* procedure takes over and reduces the threshold to half of the window size. The window size was 20 when the time-out happened so the new threshold is now 10.

TCP moves to slow start again and starts with a window size of 1, and moves to additive increase when the new threshold is reached. When the window size is 12, a three-ACKs event happens. The multiplicative decrease procedure takes over again. The threshold and window size set to 6 and TCP enters the additive increase phase this time. TCP remains in this phase until another time-out or another three-ACKs event happens.

## 15.10 TCP TIMERS

To perform its operation smoothly, most TCP implementations use at least four timers as shown in Figure 15.38.

**Figure 15.38** *TCP timers*




### Retransmission Timer

To retransmit lost segments, TCP employs one retransmission timer (for the whole connection period) that handles the retransmission time-out (RTO), the waiting time for an acknowledgment of a segment. We can define the following rules for the retransmission timer:

1. When TCP sends the segment in front of the sending queue, it starts the timer.
2. When the timer expires, TCP resends the first segment in front of the queue, and restarts the timer.
3. When a segment (or segments) are cumulatively acknowledged, the segment (or segments) are purged from the queue.
4. If the queue is empty, TCP stops the timer; otherwise, TCP restarts the timer.

#### *Round-Trip Time (RTT)*

To calculate the retransmission time-out (RTO), we first need to calculate the **round-trip time (RTT).** However, calculating RTT in TCP is an involved process that we explain step by step with some examples.

**Measured RTT** We need to find how long it takes to send a segment and receive an acknowledgment for it. This is the measured RTT. We need to remember that the segments and their acknowledgments do not have a one-to-one relationship; several segments may be acknowledged together. The measured round-trip time for a segment is the time required for the segment to reach the destination and be acknowledged, although the acknowledgment may include other segments. Note that in TCP, only one RTT measurement can be in progress at any time. This means that if an RTT measurement is started, no other measurement starts until the value of this RTT is finalized. We use the notation $RTT_M$ to stand for measured RTT.

**In TCP, there can be only one RTT measurement in progress at any time.**

**Smoothed RTT** The measured RTT, $RTT_M$, is likely to change for each round trip. The fluctuation is so high in today's Internet that a single measurement alone cannot

be used for retransmission time-out purposes. Most implementations use a smoothed RTT, called $RTT_S$, which is a weighted average of $RTT_M$ and the previous $RTT_S$ as shown below:

| | | |
|---|---|---|
| **Initially** | → | **No value** |
| **After first measurement** | → | $\mathbf{RTT_S = RTT_M}$ |
| **After each measurement** | → | $\mathbf{RTT_S = (1 - \alpha)\ RTT_S + \alpha \times RTT_M}$ |

The value of $\alpha$ is implementation-dependent, but it is normally set to 1/8. In other words, the new $RTT_S$ is calculated as 7/8 of the old $RTT_S$ and 1/8 of the current $RTT_M$.

**RTT Deviation** Most implementations do not just use $RTT_S$; they also calculate the RTT deviation, called $RTT_D$, based on the $RTT_S$ and $RTT_M$ using the following formulas:

| | | |
|---|---|---|
| **Initially** | → | **No value** |
| **After first measurement** | → | $\mathbf{RTT_D = RTT_M / 2}$ |
| **After each measurement** | → | $\mathbf{RTT_D = (1 - \beta)\ RTT_D + \beta \times |\ RTT_S - RTT_M\ |}$ |

The value of $\beta$ is also implementation-dependent, but is usually is set to 1/4.

### *Retransmission Time-out (RTO)*

The value of RTO is based on the smoothed round-trip time and its deviation. Most implementations use the following formula to calculate the RTO:

| | | |
|---|---|---|
| **Original** | → | **Initial value** |
| **After any measurement** | → | $\mathbf{RTO\ = RTT_S\ + 4 \times RTT}$ |

In other words, take the running smoothed average value of $RTT_S$, and add four times the running smoothed average value of $RTT_D$ (normally a small value).

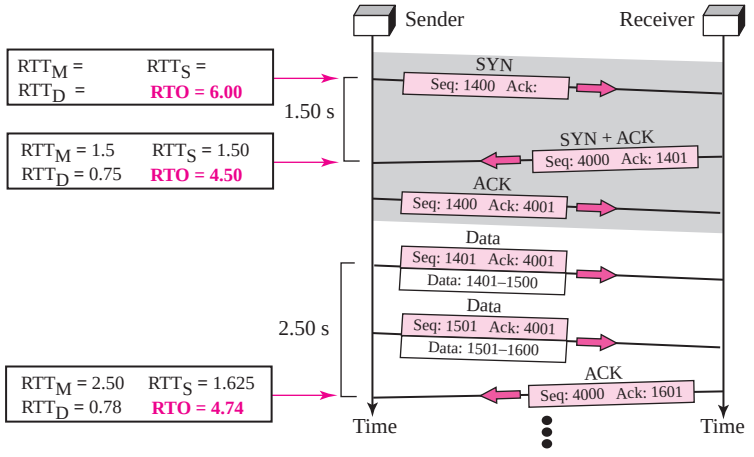
### Example 15.3

Let us give a hypothetical example. Figure 15.39 shows part of a connection. The figure shows the connection establishment and part of the data transfer phases.

1. When the SYN segment is sent, there is no value for $RTT_M$, $RTT_S$, or $RTT_D$. The value of RTO is set to 6.00 seconds. The following shows the value of these variables at this moment:

**RTO = 6**

**Figure 15.39** *Example 15.3*




2. When the SYN+ACK segment arrives, $RTT_M$ is measured and is equal to 1.5 seconds. The following shows the values of these variables:

$$\mathbf{RTT_M = 1.5}$$
$$\mathbf{RTT_S = 1.5}$$
$$\mathbf{RTT_D = (1.5) / 2 = 0.75}$$
$$\mathbf{RTO = 1.5 + 4 \times 0.75 = 4.5}$$

3. When the first data segment is sent, a new RTT measurement starts. Note that the sender does not start an RTT measurement when it sends the ACK segment, because it does not consume a sequence number and there is no time-out. No RTT measurement starts for the second data segment because a measurement is already in progress. The arrival of the last ACK segment is used to calculate the next value of $RTT_M$. Although the last ACK segment acknowledges both data segments (accumulative), its arrival finalizes the value of $RTT_M$ for the first segment. The values of these variables are now as shown below.

$$\mathbf{RTT_M = 2.5}$$
$$\mathbf{RTT_S = 7/8 \times 1.5 + (1/8) \times 2.5 = 1.625}$$
$$\mathbf{RTT_D = 3/4\ (7.5) + (1/4) \times |1.625 - 2.5| = 0.78}$$
$$\mathbf{RTO = 1.625 + 4 \times 0.78 = 4.74}$$

### *Karn's Algorithm*

Suppose that a segment is not acknowledged during the retransmission timeout period and is therefore retransmitted. When the sending TCP receives an acknowledgment for this segment, it does not know if the acknowledgment is for the original segment or for the retransmitted one. The value of the new RTT is based on the departure of the segment. However, if the original segment was lost and the acknowledgment is for the retransmitted one, the value of the current RTT must be calculated from the time the segment was retransmitted. This ambiguity was solved by Karn. **Karn's algorithm** is

simple. Do not consider the round-trip time of a retransmitted segment in the calculation of RTTs. Do not update the value of RTTs until you send a segment and receive an acknowledgment without the need for retransmission.

**TCP does not consider the RTT of a retransmitted segment in its calculation of a new RTO.**

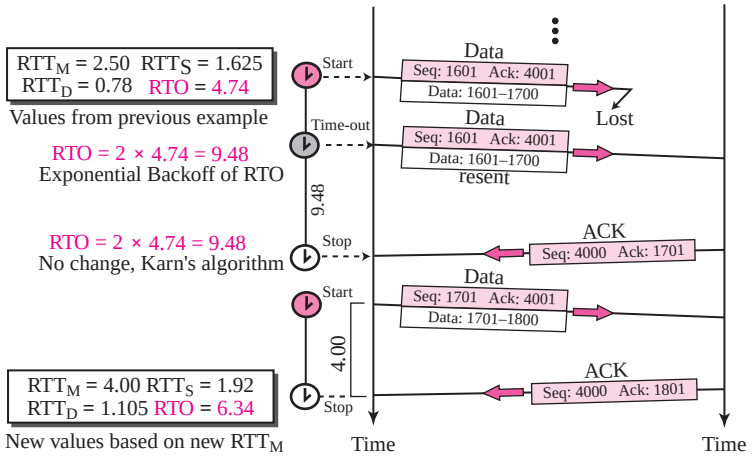
### *Exponential Backoff*

What is the value of RTO if a retransmission occurs? Most TCP implementations use an exponential backoff strategy. The value of RTO is doubled for each retransmission. So if the segment is retransmitted once, the value is two times the RTO. If it transmitted twice, the value is four times the RTO, and so on.

### Example 15.4

Figure 15.40 is a continuation of the previous example. There is retransmission and Karn's algorithm is applied.

**Figure 15.40** *Example 15.4*




The first segment in the figure is sent, but lost. The RTO timer expires after 4.74 seconds. The segment is retransmitted and the timer is set to 9.48, twice the previous value of RTO. This time an ACK is received before the time-out. We wait until we send a new segment and receive the ACK for it before recalculating the RTO (Karn's algorithm).

## Persistence Timer

To deal with a zero-window-size advertisement, TCP needs another timer. If the receiving TCP announces a window size of zero, the sending TCP stops transmitting segments until the receiving TCP sends an ACK segment announcing a nonzero window size. This ACK segment can be lost. Remember that ACK segments are not acknowledged nor retransmitted in TCP. If this acknowledgment is lost, the receiving TCP thinks that it has done its job and waits for the sending TCP to send more segments. There is no retransmission timer for

a segment containing only an acknowledgment. The sending TCP has not received an acknowledgment and waits for the other TCP to send an acknowledgment advertising the size of the window. Both TCPs might continue to wait for each other forever (a deadlock).

To correct this deadlock, TCP uses a **persistence timer** for each connection. When the sending TCP receives an acknowledgment with a window size of zero, it starts a persistence timer. When the persistence timer goes off, the sending TCP sends a special segment called a *probe*. This segment contains only 1 byte of new data. It has a sequence number, but its sequence number is never acknowledged; it is even ignored in calculating the sequence number for the rest of the data. The probe causes the receiving TCP to resend the acknowledgment.

The value of the persistence timer is set to the value of the retransmission time. However, if a response is not received from the receiver, another probe segment is sent and the value of the persistence timer is doubled and reset. The sender continues sending the probe segments and doubling and resetting the value of the persistence timer until the value reaches a threshold (usually 60 s). After that the sender sends one probe segment every 60 s until the window is reopened.

### Keepalive Timer

A **keepalive timer** is used in some implementations to prevent a long idle connection between two TCPs. Suppose that a client opens a TCP connection to a server, transfers some data, and becomes silent. Perhaps the client has crashed. In this case, the connection remains open forever.

To remedy this situation, most implementations equip a server with a keepalive timer. Each time the server hears from a client, it resets this timer. The time-out is usually 2 hours. If the server does not hear from the client after 2 hours, it sends a probe segment. If there is no response after 10 probes, each of which is 75 s apart, it assumes that the client is down and terminates the connection.

### TIME-WAIT Timer

The TIME-WAIT (2MSL) timer is used during connection termination. We discussed the reasons for this timer in Section 15.5 (State Transition Diagram).

## 15.11 OPTIONS

The TCP header can have up to 40 bytes of optional information. Options convey additional information to the destination or align other options. We can define two categories of options: 1-byte options and multiple-byte options. The first category contains two types of options: end of option list and no operation. The second category, in most implementations, contains five types of options: maximum segment size, window scale factor, timestamp, SACK-permitted, and SACK (see Figure 15.41).

#### *End of Option (EOP)*

The **end-of-option (EOP) option** is a 1-byte option used for padding at the end of the option section. It can only be used as the last option. Only one occurrence of this option is allowed. After this option, the receiver looks for the payload data. Figure 15.42 shows an example. A 3-byte option is used after the header; the data section follows this option. One EOP option is inserted to align the data with the boundary of the next word.

**Figure 15.41** *Options*

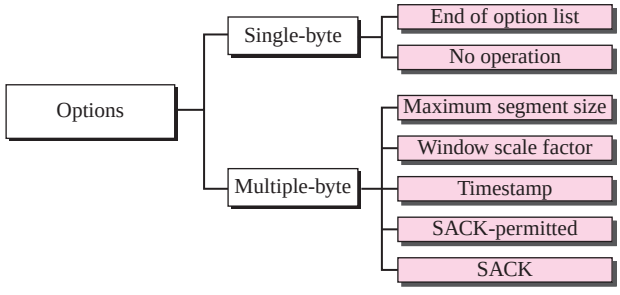
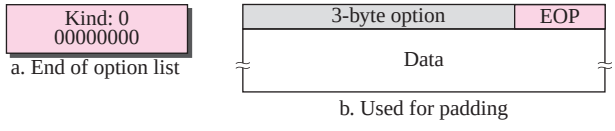



**Figure 15.42** *End-of-option*




**EOP can be used only once.**

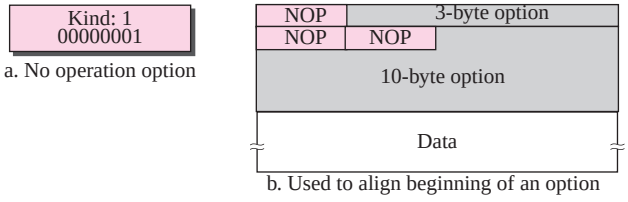
The EOP option imparts two pieces of information to the destination:

1. There are no more options in the header.
2. Data from the application program starts at the beginning of the next 32-bit word.

### *No Operation (NOP)*

The **no-operation** (**NOP**) **option** is also a 1-byte option used as a filler. However, it normally comes before another option to help align it in a four-word slot. For example, in Figure 15.43 it is used to align one 3-byte option such as the window scale factor and one 10-byte option such as the timestamp.

**Figure 15.43** *No-operation option*




**NOP can be used more than once.**

### Maximum Segment Size (MSS)

The **maximum-segment-size option** defines the size of the biggest unit of data that can be received by the destination of the TCP segment. In spite of its name, it defines the maximum size of the data, not the maximum size of the segment. Since the field is 16 bits long, the value can be 0 to 65,535 bytes. Figure 15.44 shows the format of this option.

**Figure 15.44** *Maximum-segment-size option*


MSS is determined during connection establishment. Each party defines the MSS for the segments it will receive during the connection. If a party does not define this, the default values is 536 bytes.

**The value of MSS is determined during connection establishment and does not change during the connection.**

### Window Scale Factor

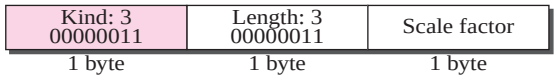
The window size field in the header defines the size of the sliding window. This field is 16 bits long, which means that the window can range from 0 to 65,535 bytes. Although this seems like a very large window size, it still may not be sufficient, especially if the data are traveling through a *long fat pipe*, a long channel with a wide bandwidth.

To increase the window size, a **window scale factor** is used. The new window size is found by first raising 2 to the number specified in the window scale factor. Then this result is multiplied by the value of the window size in the header.

$$\textbf{New window size} = \textbf{window size defined in the header} \times 2^{\textbf{window scale factor}}$$

Figure 15.45 shows the format of the window-scale-factor option.

**Figure 15.45** *Window-scale-factor option*




The scale factor is sometimes called the *shift count* because multiplying a number by a power of 2 is the same as a left shift in a bitwise operation. In other words, the actual value of the window size can be determined by taking the value of the window size advertisement in the packet and shifting it to the left in the amount of the window scale factor.

For example, suppose the value of the window scale factor is 3. An end point receives an acknowledgment in which the window size is advertised as 32,768. The size of window this end can use is $32{,}768 \times 2^3$ or 262,144 bytes. The same value can be obtained if we shift the number 32,768 three bits to the left.

Although the scale factor could be as large as 255, the largest value allowed by TCP/IP is 14, which means that the maximum window size is $2^{16} \times 2^{14} = 2^{30}$, which is less than the maximum value for the sequence number. Note that the size of the window cannot be greater than the maximum value of the sequence number.

The window scale factor can also be determined only during the connection establishment phase. During data transfer, the size of the window (specified in the header) may be changed, but it must be multiplied by the same window scale factor.

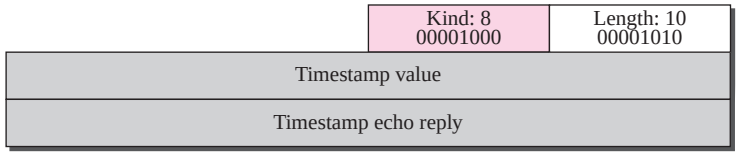
**The value of the window scale factor can be determined only during connection establishment; it does not change during the connection.**

Note that one end may set the value of the window scale factor to 0, which means that although it supports this option, it does not want to use it for this connection.

### *Timestamp*

This is a 10-byte option with the format shown in Figure 15.46. Note that the end with the active open announces a timestamp in the connection request segment (SYN segment). If it receives a timestamp in the next segment (SYN + ACK) from the other end, it is allowed to use the timestamp; otherwise, it does not use it any more. The **timestamp option** has two applications: it measures the round-trip time and prevents wrap-around sequence numbers.

**Figure 15.46** *Timestamp option*




**Measuring RTT** Timestamp can be used to measure the round-trip time (RTT). TCP, when ready to send a segment, reads the value of the system clock and inserts this value, a 32-bit number, in the timestamp value field. The receiver, when sending an acknowledgment for this segment or an accumulative acknowledgment that covers the bytes in this segment, copies the timestamp received in the timestamp echo reply. The sender, upon receiving the acknowledgment, subtracts the value of the timestamp echo reply from the time shown by the clock to find RTT.

Note that there is no need for the sender's and receiver's clocks to be synchronized because all calculations are based on the sender clock. Also note that the sender does not have to remember or store the time a segment left because this value is carried by the segment itself.

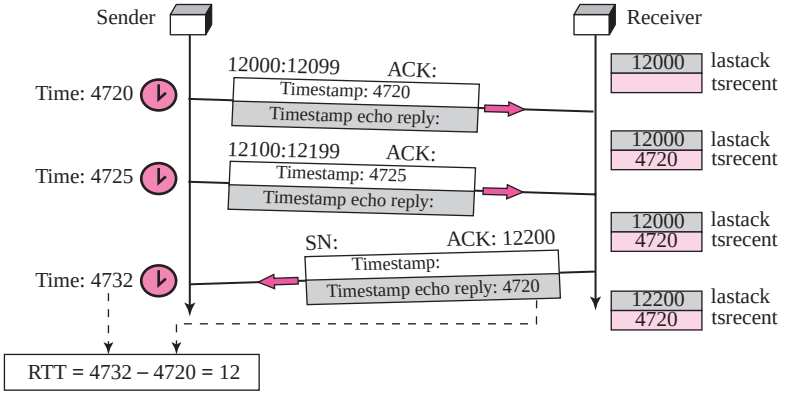
The receiver needs to keep track of two variables. The first, *lastack*, is the value of the last acknowledgment sent. The second, *tsrecent*, is the value of the recent timestamp that has not yet echoed. When the receiver receives a segment that contains the byte matching the value of lastack, it inserts the value of the timestamp field in the *tsrecent* variable. When it sends an acknowledgment, it inserts the value of tsrecent in the echo reply field.

**One application of the timestamp option is the calculation of round-trip time (RTT).**

### Example 15.5

Figure 15.47 shows an example that calculates the round-trip time for one end. Everything must be flipped if we want to calculate the RTT for the other end.

**Figure 15.47** *Example 15.5*




The sender simply inserts the value of the clock (for example, the number of seconds past midnight) in the timestamp field for the first and second segment. When an acknowledgment comes (the third segment), the value of the clock is checked and the value of the echo reply field is subtracted from the current time. RTT is 12 s in this scenario.

The receiver's function is more involved. It keeps track of the last acknowledgment sent (12000). When the first segment arrives, it contains the bytes 12000 to 12099. The first byte is the same as the value of *lastack*. It then copies the timestamp value (4720) into the *tsrecent* variable. The value of lastack is still 12000 (no new acknowledgment has been sent). When the second segment arrives, since none of the byte numbers in this segment include the value of *lastack*, the value of the timestamp field is ignored. When the receiver decides to send an accumulative acknowledgment with acknowledgment 12200, it changes the value of *lastack* to 12200 and inserts the value of tsrecent in the echo reply field. The value of tsrecent will not change until it is replaced by a new segment that carries byte 12200 (next segment).

Note that as the example shows, the RTT calculated is the time difference between sending the first segment and receiving the third segment. This is actually the meaning of RTT: the time difference between a packet sent and the acknowledgment received. The third segment carries the acknowledgment for the first and second segments.

**PAWS** The timestamp option has another application, **protection against wrapped sequence numbers (PAWS).** The sequence number defined in the TCP protocol is only 32 bits long. Although this is a large number, it could be wrapped around in a high-speed connection. This implies that if a sequence number is *n* at one time, it could be *n* again during the lifetime of the same connection. Now if the first segment is

duplicated and arrives during the second round of the sequence numbers, the segment belonging to the past is wrongly taken as the segment belonging to the new round.

One solution to this problem is to increase the size of the sequence number, but this involves increasing the size of the window as well as the format of the segment and more. The easiest solution is to include the timestamp in the identification of a segment. In other words, the identity of a segment can be defined as the combination of timestamp and sequence number. This means increasing the size of the identification. Two segments 400:12,001 and 700:12,001 definitely belong to different incarnations. The first was sent at time 400, the second at time 700.

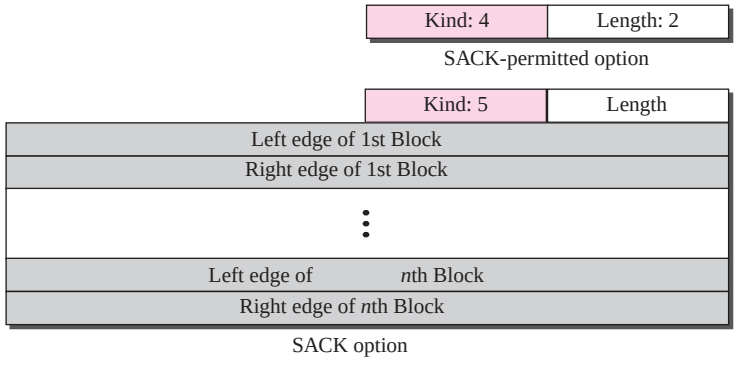
**The timestamp option can also be used for PAWS.**

### *SACK-Permitted and SACK Options*

As we discussed before, the acknowledgment field in the TCP segment is designed as an accumulative acknowledgment, which means it reports the receipt of the last consecutive byte: it does not report the bytes that have arrived out of order. It is also silent about duplicate segments. This may have a negative effect on TCP's performance. If some packets are lost or dropped, the sender must wait until a time-out and then send all packets that have not been acknowledged. The receiver may receive duplicate packets. To improve performance, selective acknowledgment (SACK) was proposed. Selective acknowledgment allows the sender to have a better idea of which segments are actually lost and which have arrived out of order. The new proposal even includes a list for duplicate packets. The sender can then send only those segments that are really lost. The list of duplicate segments can help the sender find the segments which have been retransmitted by a short time-out.

The proposal defines two new options: SACK-permitted and SACK as shown in Figure 15.48.

**Figure 15.48** *SACK*




The ***SACK*-permitted option** of two bytes is used only during connection establishment. The host that sends the SYN segment adds this option to show that it can support the SACK option. If the other end, in its SYN + ACK segment, also includes this

option, then the two ends can use the SACK option during data transfer. Note that the SACK-permitted option is not allowed during the data transfer phase.

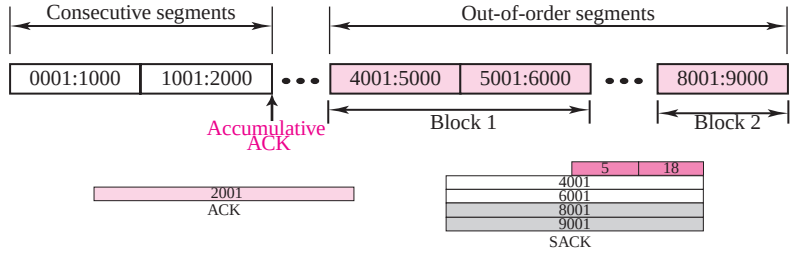
The **SACK option,** of variable length, is used during data transfer only if both ends agree (if they have exchanged SACK-permitted options during connection establishment). The option includes a list for blocks arriving out of order. Each block occupies two 32-bit numbers that define the beginning and the end of the blocks. We will show the use of this option in examples; for the moment, remember that the allowed size of an option in TCP is only 40 bytes. This means that a SACK option cannot define more than 4 blocks. The information for 5 blocks occupies $(5 \times 2) \times 4 + 2$ or 42 bytes, which is beyond the available size for the option section in a segment. If the SACK option is used with other options, then the number of blocks may be reduced.

The first block of the SACK option can be used to report the duplicates. This is used only if the implementation allows this feature.

### Example 15.6

Let us see how the SACK option is used to list out-of-order blocks. In Figure 15.49 an end has received five segments of data.

**Figure 15.49** *Example 15.6*

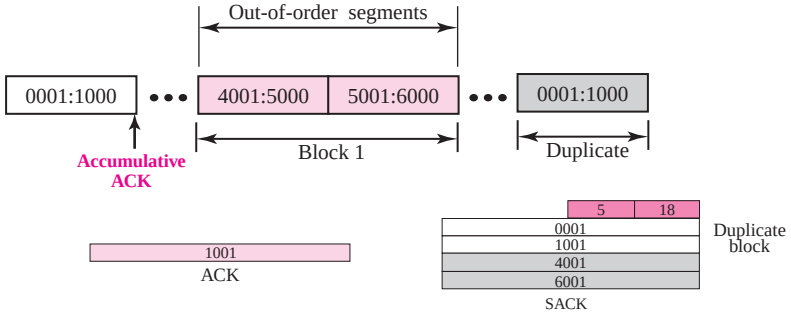



The first and second segments are in consecutive order. An accumulative acknowledgment can be sent to report the reception of these two segments. Segments 3, 4, and 5, however, are out of order with a gap between the second and third and a gap between the fourth and the fifth. An ACK and a SACK together can easily clear the situation for the sender. The value of ACK is 2001, which means that the sender need not worry about bytes 1 to 2000. The SACK has two blocks. The first block announces that bytes 4001 to 6000 have arrived out of order. The second block shows that bytes 8001 to 9000 have also arrived out of order. This means that bytes 2001 to 4000 and bytes 6001 to 8000 are lost or discarded. The sender can resend only these bytes.

### Example 15.7

Figure 15.50 shows how a duplicate segment can be detected with a combination of ACK and SACK. In this case, we have some out-of-order segments (in one block) and one duplicate segment. To show both out-of-order and duplicate data, SACK uses the first block, in this case, to show the duplicate data and other blocks to show out-of-order data. Note that only the first block can be used for duplicate data. The natural question is how the sender, when it receives these ACK and SACK values, knows that the first block is for duplicate data (compare this example with the previous example). The answer is that the bytes in the first block are already acknowledged in the ACK field; therefore, this block must be a duplicate.

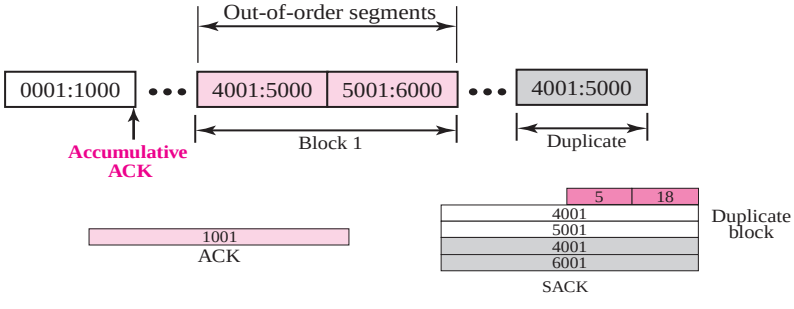
**Figure 15.50** *Example 15.7*




### Example 15.8

Figure 15.51 shows what happens if one of the segments in the out-of-order section is also duplicated. In this example, one of the segments (4001:5000) is duplicated.

**Figure 15.51** *Example 15.8*




The SACK option announces this duplicate data first and then the out-of-order block. This time, however, the duplicated block is not yet acknowledged by ACK, but because it is part of the out-of-order block (4001:5000 is part of 4001:6000), it is understood by the sender that it defines the duplicate data.

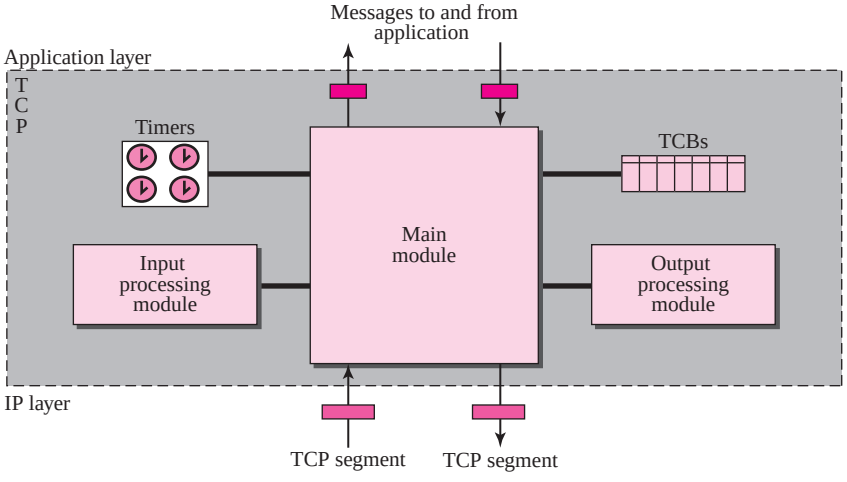
## 15.12 TCP PACKAGE

TCP is a complex protocol. It is a stream-service, connection-oriented protocol with an involved state transition diagram. It uses flow and error control. It is so complex that actual code involves tens of thousands of lines.

In this section, we present a simplified, bare-bones TCP package. Our purpose is to show how we can simulate the heart of TCP, as represented by the state transition diagram.

The package involves tables called transmission control blocks, a set of timers, and three software modules: a main module, an input processing module, and an output processing module. Figure 15.52 shows these five components and their interactions.

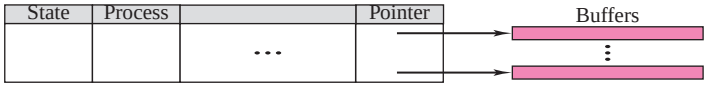
**Figure 15.52** *TCP package*




## Transmission Control Blocks (TCBs)

TCP is a connection-oriented transport protocol. A connection may be open for a long period of time. To control the connection, TCP uses a structure to hold information about each connection. This is called a *transmission control block* (TCB). Because at any time there can be several connections, TCP keeps an array of TCBs in the form of a table. The table is usually referred to as the TCB (see Figure 15.53).

**Figure 15.53** *TCBs*




Many fields can be included in each TCB. We mention only the most common ones here.

- **State.** This field defines the state of the connection according to the state transition diagram.
- **Process.** This field defines the process using this connection at this machine as a client or a server.

- **Local IP address.** This field defines the IP address of the local machine used by this connection.
- **Local port number.** This field defines the local port number used by this connection.
- **Remote IP address.** This field defines the IP address of the remote machine used by this connection.
- **Remote port number.** This field defines the remote port number used by this connection.
- **Interface.** This field defines the local interface.
- **Local window.** This field, which can comprise several subfields, holds information about the window at the local TCP.
- **Remote window.** This field, which can comprise several subfields, holds information about the window at the remote TCP.
- **Sending sequence number.** This field holds the sending sequence number.
- **Receiving sequence number.** This field holds the receiving sequence number.
- **Sending ACK number.** This field holds the value of the ACK number sent.
- **Round-trip time.** Several fields may be used to hold information about the RTT.
- **Time-out values.** Several fields can be used to hold the different time-out values such as the retransmission time-out, persistence time-out, keepalive time-out, and so on.
- **Buffer size.** This field defines the size of the buffer at the local TCP.
- **Buffer pointer.** This field is a pointer to the buffer where the received data are kept until they are read by the application.

## Timers

We have previously discussed the several timers TCP needs to keep track of its operations.

## Main Module

The main module (Table 15.3) is invoked by an arriving TCP segment, a time-out event, or a message from an application program. This is a very complicated module because the action to be taken depends on the current state of the TCP. Several approaches have been used to implement the state transition diagram including using a process for each state, using a table (two-dimensional array), and so on. To keep our discussion simple, we use cases to handle the state. We have 11 states; we use 11 different cases. Each state is implemented as defined in the state transition diagram. The **ESTABLISHED** state needs further explanation. When TCP is in this state and data or an acknowledgment segment arrives, another module, the input processing module, is called to handle the situation. Also, when TCP is in this state and a "send data" message is issued by an application program, another module, the output processing module, is called to handle the situation.

**Table 15.3** *Main Module*

```
1  TCP_Main_Module (Segment)
2  {
3       Search the TCB Table
4       if (corresponding TCB is not found)
5            Create a TCB with the state CLOSED
6       Find the state of the entry in the TCB table
7       Switch (state)
8       {
9            case CLOSED state:
10                if (“passive open” message received)
11                    go to LISTEN state.
12                if (“active open” message received)
13                {
14                    send a SYN segment
15                    go to SYN-SENT state
16                }
17                if (any segment received)
18                    send an RST segment
19                if (any other message received)
20                    issue an error message
21                break
22
23           case LISTN state:
24                if (“send data” message received)
25                {
26                    Send a SYN segment
27                    Go to SYN-SENT state
28                }
29                if (any SYN segment received)
30                {
31                    Send a SYN + ACK segment
32                    Go to SYN-RCVD state
33                }
34                if (any other segment or message received)
35                    Issue an error message
36                break
37
38           case SYN-SENT state:
```

**Table 15.3** *Main Module (continued)*

```
39                  if (time-out)
40                      Go to CLOSED state
41                  if (SYN segment received)
42                  {
43                      Send a SYN + ACK segment
44                      Go to SYN-RCVD state
45                  }
46                  if (SYN + ACK segment received)
47                  {
48                      Send an ACK segment
49                      Go to ESTABLISHED state
50                  }
51                  if (any other segment or message received)
52                      Issue an error message
53                  break
54
55              case SYN-RCVD state:
56                  if (an ACK segment received)
57                      Go to ESTABLISHID state
58                  if (time-out)
59                  {
60                      Send an RTS segment
61                      Go to CLOSED state
62                  }
63                  if (“close” message received)
64                  {
65                      Send a FIN segment
66                      Go to FIN-WAIT-I state
67                  }
68                  if (RTS segment received)
69                      Go to LISTEN state
70                  if (any other segment or message received)
71                      Issue an error message
72                  break
73
74              case ESTABLISHED state:
75                  if (a FIN segment received)
76                  {
```

**Table 15.3** *Main Module (continued)*

```
77                      Send an ACK segment
78                      Go to CLOSED-WAIT state
79                  }
80                  if ("close" message received)
81                  {
82                      Send a FIN segment
83                      Go to FIN-WAIT-I
84                  }
85                  if (a RTS or an SYN segment received)
86                      Issue an error message
87                  if (data or ACK segment received)
88                      call the input module
89                  if ("send" message received)
90                      call the output module
91                  break
92
93              case FIN-WAIT-1 state:
94                  if (a FIN segment received)
95                  {
96                      Send an ACK segment
97                      Go to CLOSING state
98                  }
99                  if (a FIN + ACK segment received)
100                 {
101                     Send an ACK segment
102                     Go to FIN-WAIT state
103                 }
104                 if (an ACK segment received)
105                     Go to FIN-WAIT-2 state
106                 if (any other segment or message received)
107                     Issue an error message
108                 break
109
110             case FIN-WAIT-2 state:
111                 if (a FIN segment received)
112                 {
113                     Send an ACK segment
114                     Go to TIME-WAIT state
115                 }
```

**Table 15.3** *Main Module (continued)*

```
116                 break
117
118             case CLOSING state:
119                 if (an ACK segment received)
120                     Go to TIME-WAIT state
121                 if (any other message or segment received)
122                     Issue an error message
123                 break
124
125             case TIME-WAIT state:
126                 if (time-out)
127                     Go to CLOSED state
128                 if (any other message or segment received)
129                     Issue an error message
130                 break
131
132             case CLOSED-WAIT state:
133                 if ("close" message received)
134                 {
135                     Send a FIN segment
136                     Go to LAST-ACK state
137                 }
138                 if (any other message or segment received)
139                     Issue an error message
140                 break
141
142             case LAST-ACK state:
143                 if (an ACK segment received)
144                     Go to CLOSED state
145                 if (any other message or segment received)
146                     Issue an error message
147                 break
148 } // end module
```

## Input Processing Module

In our design, the input processing module handles all the details needed to process data or an acknowledgment received when TCP is in the **ESTABLISHED** state. This module sends an ACK if needed, takes care of the window size announcement, does error checking, and so on. The details of this module are not needed for an introductory textbook.

### Output Processing Module

In our design, the output processing module handles all the details needed to send out data received from application program when TCP is in the **ESTABLISHED** state. This module handles retransmission time-outs, persistent time-outs, and so on. One of the ways to implement this module is to use a small transition diagram to handle different output conditions. Again, the details of this module are not needed for an introductory textbook.

## 15.13 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give a thorough coverage of TCP including [Pet & Dav 03], [Com 06], [Tan 03], and [Ste 94].

### RFCs

Several RFCs discuss TCP protocol including RFC 793, RFC 813, RFC 879, RFC 889, RFC 896, RFC 1122, RFC 1975, RFC 1987, RFC 1988, RFC 1993, RFC 2018, RFC 2581, RFC 3168, and RFC 3782.

## 15.14 KEY TERMS

Clark's solution
congestion avoidance
cookie
data transfer
deadlock
denial-of-service attack
end-of-option (EOP) option
fast retransmission
half-close
initial sequence number
Karn's algorithm
keepalive timer
maximum-segment-size option
multiplicative decrease
Nagle's algorithm
no-operation (NOP) option
persistence time
protection against wrapped sequence numbers (PAWS)
retransmission time-out (RTO)
round-trip time (RTT)
SACK option
SACK-permitted option
segment
selective acknowlegment (SACK)
silly window syndrome
simultaneous close
simultaneous open
slow start
SYN flooding attack
three-way handshaking
timestamp option
window scale factor

## 15.15 SUMMARY

- Transmission Control Protocol (TCP) is one of the transport layer protocols in the TCP/IP protocol suite. TCP provides process-to-process, full-duplex, and connection-oriented service. The unit of data transfer between two devices using TCP software is called a segment; it has 20 to 60 bytes of header, followed by data from the application program.
- A TCP connection consists of three phases: connection establishment, data transfer, and connection termination. Connection establishment requires three-way handshaking; connection termination requires three- or four-way handshaking. TCP software is normally implemented as a finite state machine (FSM).
- TCP uses flow control, implemented as a sliding window mechanism, to avoid overwhelming a receiver with data. The TCP window size is determined by the receiver-advertised window size (rwnd) or the congestion window size (cwnd), whichever is smaller. The window can be opened or closed by the receiver, but should not be shrunk. The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with a randomly generated number.
- TCP uses error control to provide a reliable service. Error control is handled by checksums, acknowledgment, and time-outs. Corrupted and lost segments are eventually retransmitted and duplicate segments are discarded. Data may arrive out of order and temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order segment is delivered to the process. In modern implementations, a retransmission occurs if the retransmission timer expires or three duplicate ACK segments have arrived.
- TCP uses congestion control to avoid and detect congestion in the network. The slow start (exponential increase), congestion avoidance (additive increase), and congestion detection (multiplicative decrease) strategies are used for congestion control. In the slow start algorithm the size of the congestion window increases exponentially until it reaches a threshold. In the congestion avoidance algorithm the size of the congestion window increases additively until congestion is detected. Different TCP implementations react differently to congestion detection. If detection is by time-out, a new slow start phase starts. If detection is by three duplicate ACKs, a new congestion avoidance phase starts.
- TCP uses four timers (retransmission, persistence, keepalive, and time-wait) in its operation. In TCP, there can be only be one RTT measurement in progress at any time. TCP does not consider the RTT of a retransmitted segment in its calculation of an RTT.
- TCP uses options to provide more services. The maximum segment size option is used in connection setup to define the largest allowable TCP segment. The value of MSS is determined during connection establishment and does not change during the connection. The window scale factor is a multiplier that increases the window size. The timestamp option shows how much time it takes for data to travel between sender and receiver. One application of the timestamp option is in the calculation of round-trip time (RTT). Another application is for PAWS. Recent implementations of

TCP use two more options, SACK-permitted option and SACK option. These two options allow the selective acknowledgment of the received segments by the receiver.

## 15.16 PRACTICE SET

### Exercises

1. Compare the TCP header and the UDP header. List the fields in the TCP header that are not part of the UDP header. Give the reason for each missing field.
2. An IP datagram is carrying a TCP segment destined for address 130.14.16.17. The destination port address is corrupted and it arrives at destination 130.14.16.19. How does the receiving TCP react to this error?
3. One ICMP message, discussed in Chapter 9, reports a destination port unreachable error. How can TCP detect the error in the destination port?
4. UDP is a message-oriented protocol. TCP is a byte-oriented protocol. If an application needs to protect the boundaries of its message, which protocol should be used, UDP or TCP?
5. What is the maximum size of the TCP header? What is the minimum size of the TCP header?
6. If the value of HLEN is 0111, how many bytes of option are included in the segment?
7. Show the entries for the header of a TCP segment that carries a message from an FTP client to an FTP server. Fill the checksum field with 0s. Choose an appropriate ephemeral port number and the correct well-known port number. The length of data is 40 bytes.
8. What can you say about the TCP segment in which the value of the control field is one of the following:
   a. 000000
   b. 000001
   c. 010001
   d. 000100
   e. 000010
   f. 010010
9. The following is a dump of a TCP header in hexadecimal format.

$$(05320017\ 00000001\ 00000000\ 500207FF\ 00000000)_{16}$$

   a. What is the source port number?
   b. What is the destination port number?
   c. What the sequence number?
   d. What is the acknowledgment number?
   e. What is the length of the header?
   f. What is the type of the segment?
   g. What is the window size?

**10.** The control field in a TCP segment is 6 bits. We can have 64 different combinations of bits. List some combinations that are valid.

**11.** To make the initial sequence number a random number, most systems start the counter at 1 during bootstrap and increment the counter by 64,000 every half second. How long does it take for the counter to wrap around?

**12.** In a TCP connection, the initial sequence number at the client site is 2,171. The client opens the connection, sends only one segment carrying 1,000 bytes of data, and closes the connection. What is the value of the sequence number in each of the following segments sent by the client?

- **a.** The SYN segment?
- **b.** The data segment?
- **c.** The FIN segment?

**13.** In a connection, the value of cwnd is 3000 and the value of rwnd is 5000. The host has sent 2,000 bytes, which have not been acknowledged. How many more bytes can be sent?

**14.** TCP opens a connection using an initial sequence number (ISN) of 14,534. The other party opens the connection with an ISN of 21,732.

- **a.** Show the three TCP segments during the connection establishment.
- **b.** Show the contents of the segments during the data transmission if the initiator sends a segment containing the message "Hello dear customer" and the other party answers with a segment containing "Hi there seller."
- **c.** Show the contents of the segments during the connection termination.

**15.** A client uses TCP to send data to a server. The data consist of 16 bytes. Calculate the efficiency of this transmission at the TCP level (ratio of useful bytes to total bytes). Calculate the efficiency of transmission at the IP level. Assume no options for the IP header. Calculate the efficiency of transmission at the data link layer. Assume no options for the IP header and use Ethernet at the data link layer.

**16.** TCP is sending data at 1 megabyte per second. If the sequence number starts with 7,000, how long does it take before the sequence number goes back to zero?

**17.** A TCP connection is using a window size of 10,000 bytes and the previous acknowledgment number was 22,001. It receives a segment with acknowledgment number 24,001 and window size advertisement of 12,000. Draw a diagram to show the situation of the window before and after.

**18.** A window holds bytes 2001 to 5000. The next byte to be sent is 3001. Draw a figure to show the situation of the window after the following two events.

- **a.** An ACK segment with the acknowledgment number 2500 and window size advertisement 4000 is received.
- **b.** A segment carrying 1,000 bytes is sent.

**19.** A TCP connection is in the **ESTABLISHED** state. The following events occur one after another:

- **a.** A FIN segment is received.
- **b.** The application sends a "close" message.

What is the state of the connection after each event? What is the action after each event?

**20.** A TCP connection is in the **ESTABLISHED** state. The following events occur one after another:

**a.** The application sends a "close" message.

**b.** An ACK segment is received.

What is the state of the connection after each event? What is the action after each event?

**21.** A host has no data to send. It receives the following segments at the times shown (hour:minute:second:milliseconds after midnight). Show the acknowledgments sent by the host.

**a.** Segment 1 received at 0:0:0:000.

**b.** Segment 2 received at 0:0:0:027.

**c.** Segment 3 received at 0:0:0:400.

**d.** Segment 4 received at 0:0:1:200.

**e.** Segment 5 received at 0:0:1:208.

**22.** A host sends five packets and receives three acknowledgments. The time is shown as hour:minute:seconds.

**a.** Segment 1 was sent at 0:0:00.

**b.** Segment 2 was sent at 0:0:05.

**c.** ACK for segments 1 and 2 received at 0:0:07.

**d.** Segment 3 was sent at 0:0:20.

**e.** Segment 4 was sent at 0:0:22.

**f.** Segment 5 was sent at 0:0:27.

**g.** ACK for segments 1 and 2 received at 0:0:45.

**h.** ACK for segment 3 received at 0:0:65.

Calculate the values of $RTT_M$, $RTT_S$, $RTT_D$, and RTO if the original RTO is 6 seconds. Did the sender miss the retransmission of any segment? Show which segments should have been retransmitted and when. Rewrite the events including the retransmission time.

**23.** Show the contents of a SACK option to be sent if a host has received bytes 2001 to 3000 in order. Bytes 4001 to 6000 are out of order, and bytes 3501 to 4000 are duplicate.

**24.** Show a congestion control diagram like Figure 15.37 using the following scenario. Assume a maximum window size of 64 segments.

**a.** Three duplicate ACKs are received after the fourth RTT.

**b.** A time-out occurs after the sixth RTT.

**25.** Show the transition diagrams (FSMs) for *simultaneous-close* scenario (See Figure 15.19).

**26.** Show the transition diagrams (FSMs) for *denying-a-connection* scenario (See Figure 15.20).

**27.** Show the transition diagrams (FSMs) for *aborting-a-connection* scenario (See Figure 15.21).

**28.** In a send window, $S_f$ = 401 and $S_n$ = 701. If window size is 1,000 bytes, show the send window before and after the station receives an ACK segment with ackNo = 601

and rwnd = 700. Ignore congestion control. Does this situation means shrinking the window?

**29.** Draw a figure similar to Figure 15.25 for the following scenario (ignore error control and congestion control):

- **a.** Time 1: The client sends a SYN segment with seqNo = 301.
- **b.** Time 2: The server sets its buffer size to 2,000 bytes.
- **c.** Time 3: The server acknowledges the SYN segment.
- **d.** Time 4: The client sends a segment of 300 bytes in the SYN + ACK segment.
- **e.** Time 5: The client sends a segment of 400 bytes.
- **f.** Time 6: The server process pulls 400 bytes.
- **g.** Time 7: The server sends an ACK.
- **h.** Time 8: The client sends a segment of 300 bytes.
- **i.** Time 9: The server process pulls 300 bytes.
- **j.** Time 10: The server sends an ACK.

**30.** Show time-line diagram for the following scenario (similar to Figure 15.29).

- **a.** The client sends a segment carrying bytes 1401 to 1700, which arrives at the sender site.
- **b.** The server sends a segment carrying bytes 2001 to 2100 and acknowledging the first segment from the client, which arrives.
- **c.** The client sends a segment carrying bytes 1701 to 1900 and acknowledging the segment received, but the segment is lost.
- **d.** The client sends a segment carrying bytes 1901 to 2100, but the segment is lost.
- **e.** Time out occurs at the client site.
- **f.** The client resends a segment in response to time-out, this packet arrived.
- **g.** The server sends an acknowledgment after ACK-delaying timer expires.
- **h.** Another time-out occurs at the client site.
- **i.** The client resends a segment in response to time-out, which arrives at the sender.
- **j.** The server sends an acknowledgment after ACK-delaying timer expires.

**31.** Redraw the time-line diagram of Figure 15.34 that allow the server to delay acknowledgements and send one ACK for each full cwnd window worth of data.

## Research Activities

**32.** We have not given all the rules about the transition diagram and TCP states. To be complete, we should show the next state for any state with the arrival of any type of segment. TCP should know what action to take if any of the segment types arrive when it is in any of the states. What are some of these rules?

**33.** What is the “half-open” case in TCP?

**34.** What is the “half-duplex close” case in TCP?

**35.** The *tcpdump* command in UNIX or LINUX can be used to print the headers of packets of a network interface. Use *tcpdump* to see the segments sent and received.

CHAPTER

# 16 Stream Control Transmission Protocol (SCTP)

In this chapter, we describe the new transport-layer protocol called SCTP. SCTP is designed as a general-purpose transport layer protocol that can handle multimedia and stream traffic, which are increasing every day on the Internet.

## OBJECTIVES

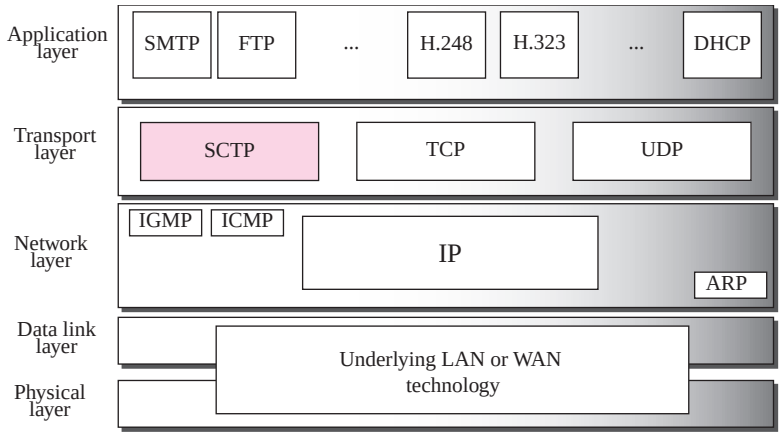
*The chapter has several objectives:*

- To introduce SCTP as a new transport-layer protocol.
- To discuss SCTP services and compare them with TCP.
- To list and explain different packet types used in SCTP and discuss the purpose and of each field in each packet.
- To discuss SCTP association and explain different scenarios such as association establishment, data transfer, association termination, and association abortion.
- To compare and contrast the state transition diagram of SCTP with the corresponding diagram of TCP.
- To explain the flow control mechanism in SCTP and discuss the behavior of the sender site and the receiver site.
- To explain the error control mechanism in SCTP and discuss the behavior of the sender site and the receiver site.
- To explain the congestion control mechanism in SCTP and compare with the similar mechanism in TCP.

## 16.1 INTRODUCTION

Stream Control Transmission Protocol (SCTP) is a new reliable, message-oriented transport-layer protocol. Figure 16.1 shows the relationship of SCTP to the other protocols in the Internet protocol suite. SCTP lies between the application layer and the network layer and serves as the intermediary between the application programs and the network operations.

**Figure 16.1** *TCP/IP protocol suite*




SCTP, however, is mostly designed for Internet applications that have recently been introduced. These new applications, such as IUA (ISDN over IP), M2UA and M3UA (telephony signaling), H.248 (media gateway control), H.323 (IP telephony), and SIP (IP telephony), need a more sophisticated service than TCP can provide. SCTP provides this enhanced performance and reliability. We briefly compare UDP, TCP, and SCTP:

- UDP is a **message-oriented** protocol. A process delivers a message to UDP, which is encapsulated in a user datagram and sent over the network. UDP *conserves the message boundaries;* each message is independent from any other message. This is a desirable feature when we are dealing with applications such as IP telephony and transmission of real-time data as we will see later in the text. However, UDP is unreliable; the sender cannot know the destiny of messages sent. A message can be lost, duplicated, or received out of order. UDP also lacks some other features, such as congestion control and flow control, needed for a friendly transport-layer protocol.

- TCP is a **byte-oriented** protocol. It receives a message or messages from a process, stores them as a stream of bytes, and sends them in segments. There is no preservation of the message boundaries. However, TCP is a reliable protocol. The duplicate segments are detected, the lost segments are resent, and the bytes are delivered to the end process in order. TCP also has congestion control and flow control mechanisms.
- SCTP combines the best features of UDP and TCP. SCTP is a reliable message-oriented protocol. It preserves the message boundaries and at the same time detects lost data, duplicate data, and out-of-order data. It also has congestion control and flow control mechanisms. Later we will see that SCTP has other innovative features unavailable in UDP and TCP.

**SCTP is a *message-oriented, reliable* protocol that combines the best features of UDP and TCP.**

## 16.2 SCTP SERVICES

Before discussing the operation of SCTP, let us explain the services offered by SCTP to the application layer processes.

### Process-to-Process Communication

SCTP uses all well-known ports in the TCP space. Table 16.1 lists some extra port numbers used by SCTP.

**Table 16.1** *Some SCTP applications*

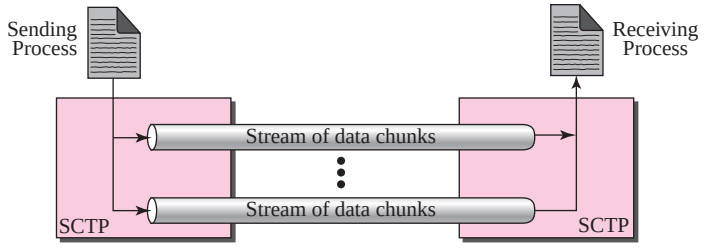
| *Protocol* | *Port Number* | *Description* |
|---|---|---|
| IUA | 9990 | ISDN over IP |
| M2UA | 2904 | SS7 telephony signaling |
| M3UA | 2905 | SS7 telephony signaling |
| H.248 | 2945 | Media gateway control |
| H.323 | 1718, 1719, 1720, 11720 | IP telephony |
| SIP | 5060 | IP telephony |

### Multiple Streams

We learned in Chapter 15 that TCP is a stream-oriented protocol. Each connection between a TCP client and a TCP server involves one single stream. The problem with this approach is that a loss at any point in the stream blocks the delivery of the rest of the data. This can be acceptable when we are transferring text; it is not when we are sending real-time data such as audio or video. SCTP allows **multistream service** in each connection, which is called **association** in SCTP terminology. If one of the streams is blocked, the other streams can still deliver their data.The idea is similar to multiple lanes on a highway. Each lane can be used for a different type of traffic. For example, one lane can be used for regular traffic, another for car pools. If the traffic is blocked for

regular vehicles, car pool vehicles can still reach their destinations. Figure 16.2 shows the idea of multiple-stream delivery.

**Figure 16.2** *Multiple-stream concept*

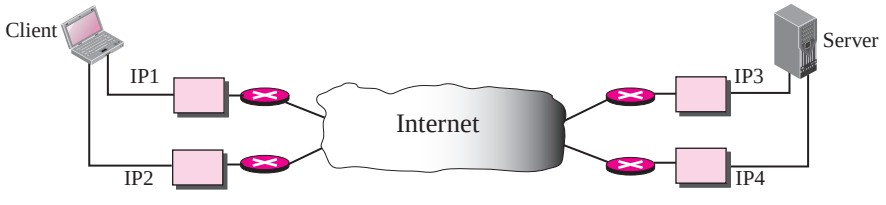



**An association in SCTP can involve multiple streams.**

## Multihoming

A TCP connection involves one source and one destination IP address. This means that even if the sender or receiver is a multihomed host (connected to more than one physical address with multiple IP addresses), only one of these IP addresses per end can be utilized during the connection. An SCTP association, on the other hand, supports **multihoming service.** The sending and receiving host can define multiple IP addresses in each end for an association. In this fault-tolerant approach, when one path fails, another interface can be used for data delivery without interruption. This fault-tolerant feature is very helpful when we are sending and receiving a real-time payload such as Internet telephony. Figure 16.3 shows the idea of multihoming.

**Figure 16.3** *Multihoming concept*




In the figure, the client is connected to two local networks with two IP addresses. The server is also connected to two networks with two IP addresses. The client and the server can make an association using four different pairs of IP addresses. However, note that in the current implementations of SCTP, only one pair of IP addresses can be chosen for normal communication; the alternative is used if the main choice fails. In other words, at present, SCTP does not allow load sharing between different paths.

**SCTP association allows multiple IP addresses for each end.**

### Full-Duplex Communication

Like TCP, SCTP offers **full-duplex service,** where data can flow in both directions at the same time. Each SCTP then has a sending and receiving buffer and packets are sent in both directions.

### Connection-Oriented Service

Like TCP, SCTP is a connection-oriented protocol. However, in SCTP, a connection is called an **association.** When a process at site A wants to send and receive data from another process at site B, the following occurs:

1. The two SCTPs establish an association between each other.
2. Data are exchanged in both directions.
3. The association is terminated.

### Reliable Service

SCTP, like TCP, is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data. We will discuss this feature further in the section on error control.

## 16.3 SCTP FEATURES

Let us first discuss the general features of SCTP and then compare them with those of TCP.

### Transmission Sequence Number (TSN)

The unit of data in TCP is a byte. Data transfer in TCP is controlled by numbering bytes using a sequence number. On the other hand, the unit of data in SCTP is a data chunk, which may or may not have a one-to-one relationship with the message coming from the process because of fragmentation (discussed later). Data transfer in SCTP is controlled by numbering the data chunks. SCTP uses a **transmission sequence number (TSN)** to number the data chunks. In other words, the TSN in SCTP plays the analogous role as the sequence number in TCP. TSNs are 32 bits long and randomly initialized between 0 and $2^{32} - 1$. Each data chunk must carry the corresponding TSN in its header.

**In SCTP, a data chunk is numbered using a TSN.**

### Stream Identifier (SI)

In TCP, there is only one stream in each connection. In SCTP, there may be several streams in each association. Each stream in SCTP needs to be identified using a **stream identifier (SI).** Each data chunk must carry the SI in its header so that when it arrives at the destination, it can be properly placed in its stream. The SI is a 16-bit number starting from 0.

**To distinguish between different streams, SCTP uses an SI.**

## Stream Sequence Number (SSN)

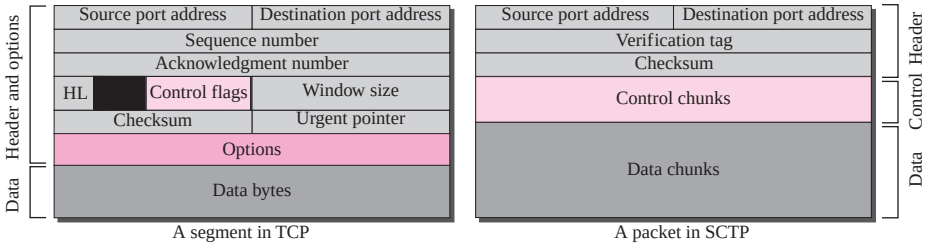
When a data chunk arrives at the destination SCTP, it is delivered to the appropriate stream and in the proper order. This means that, in addition to an SI, SCTP defines each data chunk in each stream with a **stream sequence number (SSN).**

**To distinguish between different data chunks belonging to the same stream, SCTP uses SSNs.**

## Packets

In TCP, a segment carries data and control information. Data are carried as a collection of bytes; control information is defined by six control flags in the header. The design of SCTP is totally different: data are carried as data chunks, control information as control chunks. Several control chunks and data chunks can be packed together in a packet. A packet in SCTP plays the same role as a segment in TCP. Figure 16.4 compares a segment in TCP and a packet in SCTP.

**Figure 16.4** *Comparison between a TCP segment and an SCTP packet*




**TCP has segments; SCTP has packets.**

We will discuss the format of the SCTP packet in the next section. For the moment, let us briefly list the differences between an SCTP packet and a TCP segment:

1. The control information in TCP is part of the header; the control information in SCTP is included in the control chunks. There are several types of control chunks; each is used for a different purpose.
2. The data in a TCP segment treated as one entity; an SCTP packet can carry several data chunks; each can belong to a different stream.
3. The options section, which can be part of a TCP segment, does not exist in an SCTP packet. Options in SCTP are handled by defining new chunk types.
4. The mandatory part of the TCP header is 20 bytes, while the general header in SCTP is only 12 bytes. The SCTP header is shorter due to the following:
   a. An SCTP sequence number (TSN) belongs to each data chunk, and hence is located in the chunk's header.

b. The acknowledgment number and window size are part of each control chunk.
c. There is no need for a header length field (shown as HL in the TCP segment) because there are no options to make the length of the header variable; the SCTP header length is fixed (12 bytes).
d. There is no need for an urgent pointer in SCTP, as we will see later.

5. The checksum in TCP is 16 bits; in SCTP, it is 32 bits.
6. The verification tag in SCTP is an association identifier, which does not exist in TCP. In TCP, the combination of IP and port addresses define a connection; in SCTP we may have multihoming using different IP addresses. A unique verification tag is needed to define each association.
7. TCP includes one sequence number in the header, which defines the number of the first byte in the data section. An SCTP packet can include several different data chunks. TSNs, ISs, and SSNs define each data chunk.
8. Some segments in TCP that carry control information (such as SYN and FIN), need to consume one sequence number; control chunks in SCTP never use a TSN, IS, or SSN number. These three identifiers belong only to data chunks, not to the whole packet.

**In SCTP, control information and data information are carried in separate chunks.**

In SCTP, we have data chunks, streams, and packets. An association may send many packets, a packet may contain several chunks, and chunks may belong to different streams. To make the definitions of these terms clear, let us suppose that process A needs to send 11 messages to process B in three streams. The first four messages are in the first stream, the second three messages are in the second stream, and the last four messages are in the third stream.

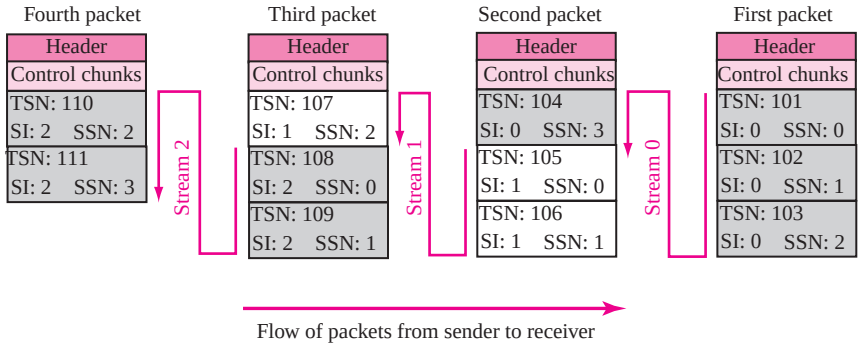
Although a message, if long, can be carried by several data chunks, we assume that each message fits into one data chunk. Therefore, we have 11 data chunks in three streams.

The application process delivers 11 messages to SCTP, where each message is earmarked for the appropriate stream. Although the process could deliver one message from the first stream and then another from the second, we assume that it delivers all messages belonging to the first stream first, all messages belonging to the second stream next, and finally, all messages belonging to the last stream.

We also assume that the network allows only 3 data chunks per packet, which means that we need 4 packets as shown in Figure 16.5. Data chunks in stream 0 are carried in the first and part of the second packet; those in stream 1 are carried in the second and the third packet; those in stream 2 are carried in the third and fourth packet.

Note that each data chunk needs three identifiers: TSN, SI, and SSN. TSN is a cumulative number and used, as we will see later, for flow control and error control. SI defines the stream to which the chunk belongs. SSN defines the chunk's order in a particular stream. In our example, SSN starts from 0 for each stream.

**Figure 16.5** *Packet, data chunks, and streams*




**Data chunks are identified by three identifiers: TSN, SI, and SSN. TSN is a cumulative number identifying the association; SI defines the stream; SSN defines the chunk in a stream.**

## Acknowledgment Number

TCP acknowledgment numbers are byte-oriented and refer to the sequence numbers. SCTP acknowledgment numbers are chunk-oriented. They refer to the TSN. A second difference between TCP and SCTP acknowledgments is the control information. Recall that this information is part of the segment header in TCP. To acknowledge segments that carry only control information, TCP uses a sequence number and acknowledgment number (for example, a SYN segment needs to be acknowledged by an ACK segment). In SCTP, however, the control information is carried by control chunks, which do not need a TSN. These control chunks are acknowledged by another control chunk of the appropriate type (some need no acknowledgment). For example, an INIT control chunk is acknowledged by an INIT-ACK chunk. There is no need for a sequence number or an acknowledgment number.

**In SCTP, acknowledgment numbers are used to acknowledge only data chunks; control chunks are acknowledged by other control chunks if necessary.**

## Flow Control

Like TCP, SCTP implements flow control to avoid overwhelming the receiver. We will discuss SCTP flow control later in the chapter.

## Error Control

Like TCP, SCTP implements error control to provide reliability. TSN numbers and acknowledgment numbers are used for error control. We will discuss error control later in the chapter.

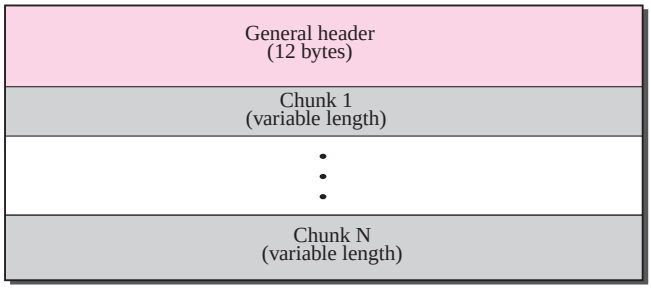
### Congestion Control

Like TCP, SCTP implements congestion control to determine how many data chunks can be injected into the network. We will discuss congestion control later in the chapter.

## 16.4 PACKET FORMAT

In this section, we show the format of a packet and different types of chunks. Most of the information presented in this section will become clear later; this section can be skipped in the first reading or used only as the reference. An SCTP packet has a mandatory general header and a set of blocks called chunks. There are two types of chunks: control chunks and data chunks. A control chunk controls and maintains the association; a data chunk carries user data. In a packet, the control chunks come before the data chunks. Figure 16.6 shows the general format of an SCTP packet.

**Figure 16.6** *SCTP packet format*

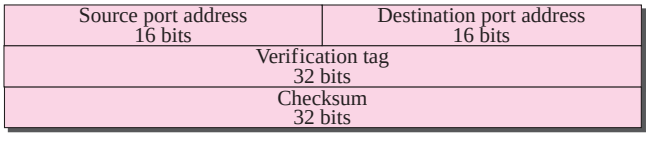



**In an SCTP packet, control chunks come before data chunks.**

### General Header

The **general header** (packet header) defines the end points of each association to which the packet belongs, guarantees that the packet belongs to a particular association, and preserves the integrity of the contents of the packet including the header itself. The format of the general header is shown in Figure 16.7.

**Figure 16.7** *General header*



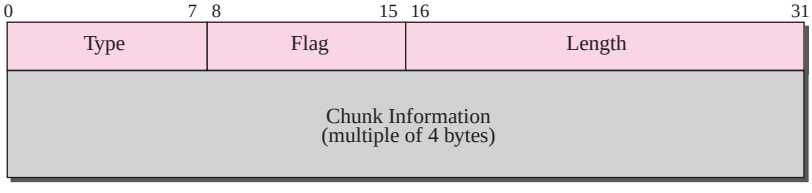
There are four fields in the general header:

- **Source port address.** This is a 16-bit field that defines the port number of the process sending the packet.
- **Destination port address.** This is a 16-bit field that defines the port number of the process receiving the packet.
- **Verification tag.** This is a number that matches a packet to an association. This prevents a packet from a previous association from being mistaken as a packet in this association. It serves as an identifier for the association; it is repeated in every packet during the association. There is a separate verification used for each direction in the association.
- **Checksum.** This 32-bit field contains a CRC-32 checksum (see Appendix D). Note that the size of the checksum is increased from 16 bits (in UDP, TCP, and IP) to 32 bits in SCTP to allow the use of the CRC-32 checksum.

## Chunks

Control information or user data are carried in chunks. Chunks have a common layout as shown in Figure 16.8.

**Figure 16.8** *Common layout of a chunk*




The first three fields are common to all chunks; the information field depends on the type of chunk. The important point to remember is that SCTP requires the information section to be a multiple of 4 bytes; if not, padding bytes (eight 0s) are added at the end of the section.

**Chunks need to terminate on a 32-bit (4-byte) boundary.**

The description of the common fields are as follows:

- **Type.** This 8-bit field can define up to 256 types of chunks. Only a few have been defined so far; the rest are reserved for future use. See Table 16.2 for a list of chunks and their descriptions.
- **Flag.** This 8-bit field defines special flags that a particular chunk may need. Each bit has a different meaning depending on the type of chunk.
- **Length.** Since the size of the information section is dependent on the type of chunk, we need to define the chunk boundaries. This 16-bit field defines the total

**Table 16.2** *Chunks*

| *Type* | *Chunk* | *Description* |
|---|---|---|
| 0 | DATA | User data |
| 1 | INIT | Sets up an association |
| 2 | INIT ACK | Acknowledges INIT chunk |
| 3 | SACK | Selective acknowledgment |
| 4 | HEARTBEAT | Probes the peer for liveliness |
| 5 | HEARTBEAT ACK | Acknowledges HEARTBEAT chunk |
| 6 | ABORT | Abort an association |
| 7 | SHUTDOWN | Terminates an association |
| 8 | SHUTDOWN ACK | Acknowledges SHUTDOWN chunk |
| 9 | ERROR | Reports errors without shutting down |
| 10 | COOKIE ECHO | Third packet in association establishment |
| 11 | COOKIE ACK | Acknowledges COOKIE ECHO chunk |
| 14 | SHUTDOWN COMPLETE | Third packet in association termination |
| 192 | FORWARD TSN | For adjusting cumulating TSN |

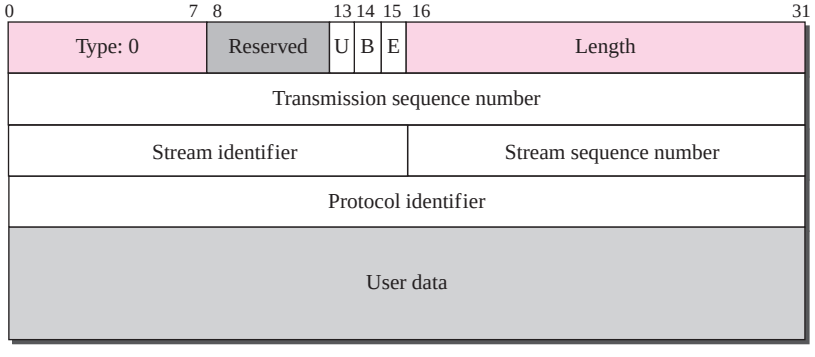
size of the chunk, in bytes, including the type, flag, and length fields. If a chunk carries no information, the value of the length field is 4 (4 bytes). Note that the length of the padding, if any, is not included in the calculation of the length field. This helps the receiver find out how many useful bytes a chunk carries. If the value is not a multiple of 4, the receiver knows there is padding. For example, when the receiver sees a length of 17, it knows the next number that is a multiple of 4 is 20, so there are 3 bytes of padding that must be discarded. But if the receiver sees a length of 16, it knows that there is no padding.

**The number of padding bytes is not included in the value of the length field.**

### *DATA*

The **DATA chunk** carries the user data. A packet may contain zero or more data chunks. Figure 16.9 shows the format of a DATA chunk.

**Figure 16.9** *DATA chunk*



The descriptions of the common fields are the same. The type field has a value of 0. The flag field has 5 reserved bits and 3 defined bits: U, B, and E. The U (unordered) field, when set to 1, signals unordered data (explained later). In this case, the value of the stream sequence number is ignored. The B (beginning) and E (end) bits together define the position of a chunk in a message that is fragmented. When B = 1 and E = 1, there is no fragmentation (first and last); the whole message is carried in one chunk. When B = 1 and E = 0, it is the first fragment. When B = 0 and E = 1, it is the last fragment. When B = 0 and E = 0, it is a middle fragment (neither the first nor the last). Note that the value of the length field does not include padding. This value cannot be less than 17 because a DATA chunk must always carry at least one byte of data.

- **Transmission sequence number (TSN).** This 32-bit field defines the transmission sequence number. It is a sequence number that is initialized in an INIT chunk for one direction and in the INIT ACK chunk for the opposite direction.
- **Stream identifier (SI).** This 16-bit field defines each stream in an association. All chunks belonging to the same stream in one direction carry the same stream identifier.
- **Stream sequence number (SSN).** This 16-bit field defines a chunk in a particular stream in one direction.
- **Protocol identifier.** This 32-bit field can be used by the application program to define the type of data. It is ignored by the SCTP layer.
- **User data.** This field carries the actual user data. SCTP has some specific rules about the user data field. First, no chunk can carry data belonging to more than one message, but a message can be spread over several data chunks. Second, this field cannot be empty; it must have at least one byte of user data. Third, if the data cannot end at a 32-bit boundary, padding must be added. These padding bytes are not included in the value of the length field.

**A DATA chunk cannot carry data belonging to more than one message, but a message can be split into several chunks. The data field of the DATA chunk must carry at least one byte of data, which means the value of length field cannot be less than 17.**

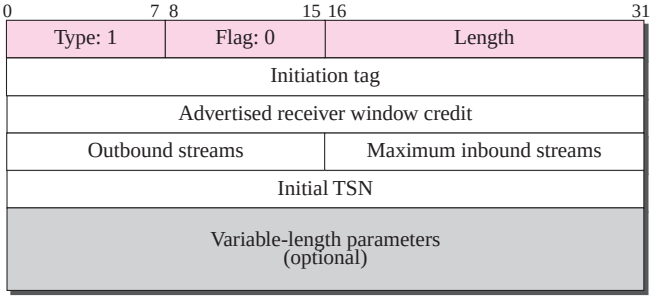
### *INIT*

The **INIT chunk** (initiation chunk) is the first chunk sent by an end point to establish an association. The packet that carries this chunk cannot carry any other control or data chunks. The value of the verification tag for this packet is 0, which means no tag has yet been defined. The format is shown in Figure 16.10.

The three common fields (type, flag, and length) are as before. The value of the type field is 1. The value of the flag field is zero (no flags); and the value of the length field is a minimum of 20 (more if there are optional parameters). The other fields are explained below:

- **Initiation tag.** This 32-bit field defines the value of the verification tag for packets traveling in the opposite direction. As we mentioned before, all packets have a verification tag in the general header; this tag is the same for all packets traveling in one direction in an association. The value of this tag is determined during

**Figure 16.10** *INIT chunk*




association establishment. The end point that initiates the association defines the value of this tag in the initiation tag field. This value is used as the verification tag in the rest of the packets sent from the other direction. For example, when end point A starts an association with end point B, A defines an initiation tag value, say *x*, which is used as the verification tag for all packets sent from B to A. The initiation tag is a random number between 0 and $2^{32} - 1$. The value of 0 defines no association and is permitted only by the general header of the INIT chunk.

- **Advertised receiver window credit.** This 32-bit field is used in flow control and defines the initial amount of data in bytes that the sender of the INIT chunk can allow. It is the rwnd value that will be used by the receiver to know how much data to send. Note that, in SCTP, sequence numbers are in terms of chunks.
- **Outbound stream.** This 16-bit field defines the number of streams that the initiator of the association suggests for streams in the outbound direction. It may be reduced by the other end point.
- **Maximum inbound stream.** This 16-bit field defines the maximum number of streams that the initiator of the association can support in the inbound direction. Note that this is a maximum number and cannot be increased by the other end point.
- **Initial TSN.** This 32-bit field initializes the transmission sequence number (TSN) in the outbound direction. Note that each data chunk in an association has to have one TSN. The value of this field is also a random number less than $2^{32}$.
- **Variable-length parameters.** These optional parameters may be added to the INIT chunk to define the IP address of sending end point, the number of IP addresses the end point can support (multihome), the preservation of the cookie state, the type of addresses, and support of explicit congestion notification (ECN).

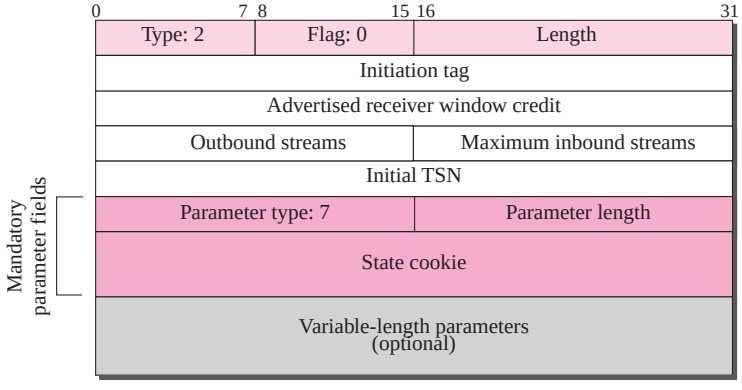
**No other chunk can be carried in a packet that carries an INIT chunk.**

### *INIT ACK*

The **INIT ACK chunk (**initiation acknowledgment chunk**)** is the second chunk sent during association establishment. The packet that carries this chunk cannot carry any

other control or data chunks. The value of the verification tag for this packet (located in the general header) is the value of the initiation tag defined in the received INIT chunk. The format is shown in Figure 16.11.

**Figure 16.11** *INIT ACK chunk*



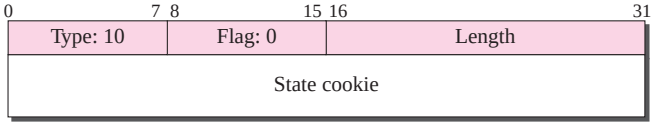

Note that the fields in the main part of the chunk are the same as those defined in the INIT chunk. However, a mandatory parameter is required for this chunk. The parameter of type 7 defines the state cookie sent by the sender of this chunk. We discuss the use of cookies later in the chapter. The chunk can also have optional parameters. Note that the initiation tag field in this chunk initiates the value of the verification tag for future packets traveling from the opposite direction.

**No other chunk can be carried in a packet that carries an INIT ACK chunk.**

### *COOKIE ECHO*

The **COOKIE ECHO chunk** is the third chunk sent during association establishment. It is sent by the end point that receives an INIT ACK chunk (normally the sender of the INIT chunk). The packet that carries this chunk can also carry user data. The format is shown in Figure 16.12.

**Figure 16.12** *COOKIE ECHO chunk*

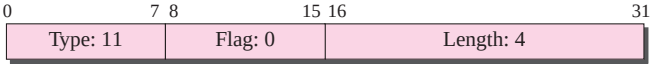



Note that this is a very simple chunk of type 10. In the information section it echoes the state cookie that the end point has previously received in the INIT ACK. The receiver of the INIT ACK cannot open the cookie.

### *COOKIE ACK*

The **COOKIE ACK chunk** is the fourth and last chunk sent during association establishment. It is sent by an end point that receives a COOKIE ECHO chunk. The packet that carries this chunk can also carry user data. The format is shown in Figure 16.13.

**Figure 16.13** *COOKIE ACK*

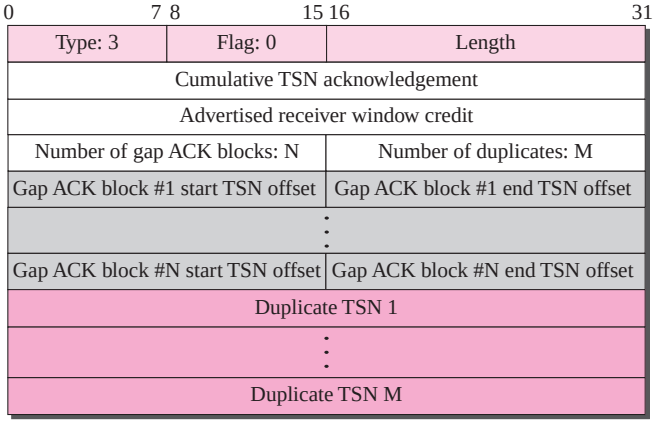



Note that this is a very simple chunk of type 11. The length of the chunk is exactly 4 bytes.

### *SACK*

The **SACK chunk** (selective ACK chunk) acknowledges the receipt of data packets. Figure 16.14 shows the format of the SACK chunk.

**Figure 16.14** *SACK chunk*




The common fields are the same as discussed previously. The type field has a value of 3. The flag bits are all set to 0s.

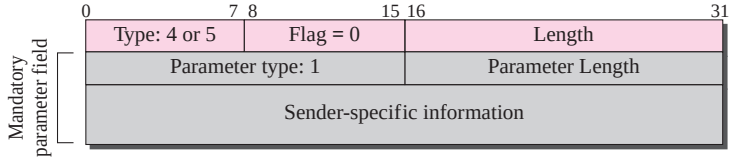
- **Cumulative TSN acknowledgment.** This 32-bit field defines the TSN of the last data chunk received in sequence.
- **Advertised receiver window credit.** This 32-bit field is the updated value for the receiver window size.
- **Number of gap ACK blocks.** This 16-bit field defines the number of gaps in the data chunk received after the cumulative TSN. Note that the term *gap* is misleading here: the gap defines the sequence of received chunks, not the missing chunks.
- **Number of duplicates.** This 16-bit field defines the number of duplicate chunks following the cumulative TSN.

- **Gap ACK block start offset.** For each gap block, this 16-bit field gives the starting TSN relative to the cumulative TSN.
- **Gap ACK block end offset.** For each gap block, this 16-bit field gives the ending TSN relative to the cumulative TSN.
- **Duplicate TSN.** For each duplicate chunk, this 32-bit field gives the TSN of the duplicate chunk.

### *HEARTBEAT and HEARTBEAT ACK*

The **HEARTBEAT chunk** and **HEARTBEAT ACK chunk** are similar except for the type field. The first has a type of 4 and the second a type of 5. Figure 16.15 shows the format of these chunks. These two chunks are used to periodically probe the condition of an association. An end point sends a HEARTBEAT chunk; the peer responds with a HEARTBEAT ACK if it is alive. The format has the common three fields and mandatory parameter fields that provide sender-specific information. This information in the HEARTBEAT chunk includes the local time and the address of the sender. It is copied without change into the HEARTBEAT ACK chunk.

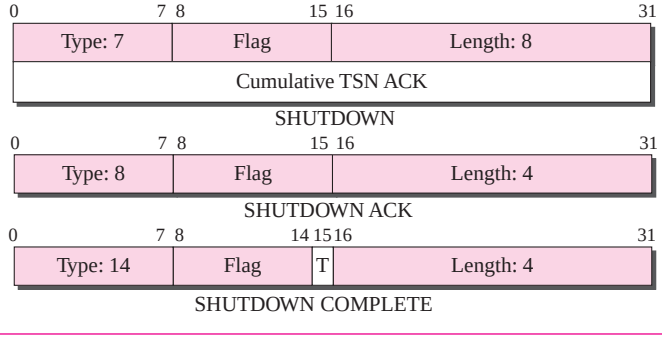
**Figure 16.15** *HEARTBEAT and HEARTBEAT ACK chunks*




### *SHUTDOWN, SHUTDOWN ACK, and SHUTDOWN COMPLETE*

These three chunks (used for closing an association) are similar. The **SHUTDOWN chunk,** type 7, is eight bytes in length; the second four bytes define the cumulative TSN. The **SHUTDOWN ACK chunk,** type 8, is four bytes in length. The **SHUTDOWN COMPLETE chunk,** type 14, is also 4 bytes long, and has a one bit flag, the T flag. The T flag shows that the sender does not have a TCB table (see Chapter 15). Figure 16.16 shows the formats.

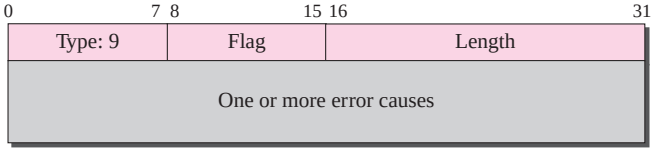
**Figure 16.16** *SHUTDOWN, SHUTDOWN ACK, and SHUTDOWN COMPLETE chunks*



### ERROR

The **ERROR chunk** is sent when an end point finds some error in a received packet. Note that the sending of an ERROR chunk does not imply the aborting of the association. (This would require an ABORT chunk.) Figure 16.17 shows the format of the ERROR chunk.

**Figure 16.17** *ERROR chunk*




The errors are defined in Table 16.3.

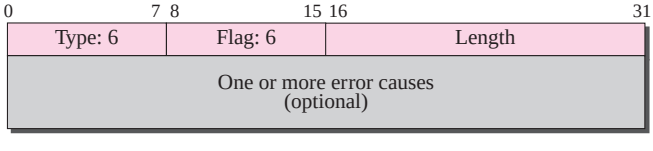
**Table 16.3** *Errors*

| *Code* | *Description* |
|---|---|
| 1 | Invalid stream identifier |
| 2 | Missing mandatory parameter |
| 3 | State cookie error |
| 4 | Out of resource |
| 5 | Unresolvable address |
| 6 | Unrecognized chunk type |
| 7 | Invalid mandatory parameters |
| 8 | Unrecognized parameter |
| 9 | No user data |
| 10 | Cookie received while shutting down |

### ABORT

The **ABORT chunk** is sent when an end point finds a fatal error and needs to abort the association. The error types are the same as those for the ERROR chunk (see Table 16.3). Figure 16.18 shows the format of an ABORT chunk.

**Figure 16.18** *ABORT chunk*




### FORWARD TSN

This is a chunk recently added to the standard (see RFC 3758) to inform the receiver to adjust its cumulative TSN. It provides partial reliable service.

## 16.5 AN SCTP ASSOCIATION

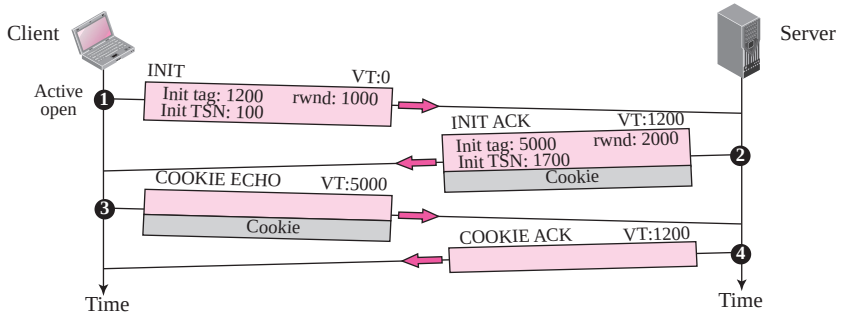
SCTP, like TCP, is a connection-oriented protocol. However, a connection in SCTP is called an *association* to emphasize multihoming.

**A connection in SCTP is called an association.**

### Association Establishment

**Association establishment** in SCTP requires a *four-way handshake*. In this procedure, a process, normally a client, wants to establish an association with another process, normally a server, using SCTP as the transport layer protocol. Similar to TCP, the SCTP server needs to be prepared to receive any association (passive open). Association establishment, however, is initiated by the client (active open). SCTP association establishment is shown in Figure 16.19.

**Figure 16.19** *Four-way handshaking*




The steps, in a normal situation, are as follows:

1. The client sends the first packet, which contains an INIT chunk. The **verification tag** (VT) of this packet (defined in the general header) is 0 because no verification tag has yet been defined for this direction (client to server).The INIT tag includes an initiation tag to be used for packets from the other direction (server to client). The chunk also defines the initial TSN for this direction and advertises a value for rwnd. The value of rwnd is normally advertised in a SACK chunk; it is done here because SCTP allows the inclusion of a DATA chunk in the third and fourth packets; the server must be aware of the available client buffer size. Note that no other chunks can be sent with the first packet.
2. The server sends the second packet, which contains an INIT ACK chunk. The verification tag is the value of the initial tag field in the INIT chunk. This chunk initiates the tag to be used in the other direction, defines the initial TSN, for data flow from server to client, and sets the servers' rwnd. The value of rwnd is defined to allow the client to send a DATA chunk with the third packet. The INIT ACK also

sends a cookie that defines the state of the server at this moment. We will discuss the use of the cookie shortly.

3. The client sends the third packet, which includes a COOKIE ECHO chunk. This is a very simple chunk that echoes, without change, the cookie sent by the server. SCTP allows the inclusion of data chunks in this packet.
4. The server sends the fourth packet, which includes the COOKIE ACK chunk that acknowledges the receipt of the COOKIE ECHO chunk. SCTP allows the inclusion of data chunks with this packet.

**No other chunk is allowed in a packet carrying an INIT or INIT ACK chunk. A COOKIE ECHO or a COOKIE ACK chunk can carry data chunks.**

### *Number of Packets Exchanged*

The number of packets exchanged is three in a TCP connection establishment, and four in an SCTP association establishment. It might seem that SCTP is less efficient than TCP, but we need to consider that SCTP allows the exchange of data in the third and fourth packets and, as we shall see, provides better security against SYN denial-of-service attacks. After two packets are exchanged, data can be transferred.

### *Verification Tag*

When we compare TCP and SCTP, we find that the verification tag in SCTP does not exist in TCP. In TCP, a connection is identified by a combination of IP addresses and port numbers which is part of each segment. This has created two problems:

1. A blind attacker (not an interceptor) can send segments to a TCP server using randomly chosen source and destination port numbers such as those we discussed in a SYN flooding attack.
2. A delayed segment from a previous connection can show up in a new connection that uses the same source and destination port addresses (incarnation). This was one of the reasons that TCP needs a TIME-WAIT timer when terminating a connection.

SCTP solves these two problems by using a verification tag, a common value that is carried in all packets traveling in one direction in an association. A blind attacker cannot inject a random packet into an association because the packet would most likely not carry the appropriate tag (odds are 1 out of $2^{32}$). A packet from an old association cannot show up in an incarnation because, even if the source and destination port addresses are the same, the verification tag would surely be different. Two verification tags, one for each direction, identify an association.

### *Cookie*

We discussed a SYN flooding attack in Chapter 15. With TCP, a malicious attacker can flood a TCP server with a huge number of phony SYN segments using different forged IP addresses. Each time the server receives a SYN segment, it sets up a TCB and allocates other resources while waiting for the next segment to arrive. After a while, however, the server may collapse due to the exhaustion of resources.

The designers of SCTP have a strategy to prevent this type of attack. The strategy is to postpone the allocation of resources until the reception of the third packet, when the IP address of the sender is verified. The information received in the first packet must somehow be saved until the third packet arrives. But if the server saves the information, that would require the allocation of resources (memory); this is the dilemma. The solution is to pack the information and send it back to the client. This is called generating a **cookie.** The cookie is sent with the second packet to the address received in the first packet. There are two potential situations.

1. If the sender of the first packet is an attacker, the server never receives the third packet; the cookie is lost and no resources are allocated. The only effort for the server is "baking" the cookie.
2. If the sender of the first packet is an honest client that needs to make a connection, it receives the second packet, with the cookie. It sends a packet (third in the series) with the cookie, with no changes. The server receives the third packet and knows that it has come from an honest client because the cookie that the sender has sent is there. The server can now allocate resources.

The above strategy works if no entity can "eat" a cookie "baked" by the server. To guarantee this, the server creates a digest (see Chapter 29) from the information using its own secret key. The information and the digest, together make the cookie, which is sent to the client in the second packet. When the cookie is returned in the third packet, the server calculates the digest from the information. If the digest matches the one that is sent, the cookie has not been changed by any other entity.

## Data Transfer

The whole purpose of an association is to transfer data between two ends. After the association is established, bidirectional data transfer can take place. The client and the server can both send data. Like TCP, SCTP supports piggybacking.

There is a major difference, however, between data transfer in TCP and SCTP. TCP receives messages from a process as a stream of bytes without recognizing any boundary between them. The process may insert some boundaries for its peer use, but TCP treats that mark as part of the text. In other words, TCP takes each message and appends it to its buffer. A segment can carry parts of two different messages. The only ordering system imposed by TCP is the byte numbers.

SCTP, on the other hand, recognizes and maintains boundaries. Each message coming from the process is treated as one unit and inserted into a DATA chunk unless it is fragmented (discussed later). In this sense, SCTP is like UDP, with one big advantage: data chunks are related to each other.

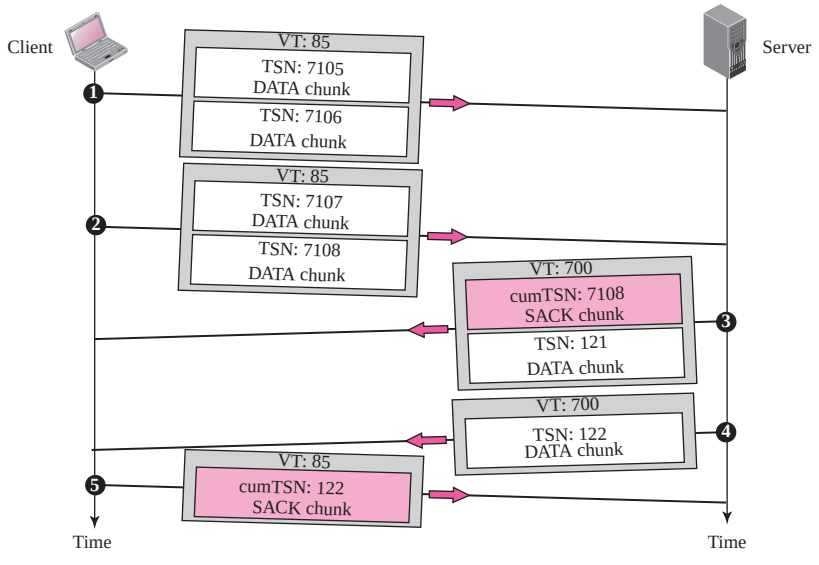
A message received from a process becomes a DATA chunk, or chunks if fragmented, by adding a DATA chunk header to the message. Each DATA chunk formed by a message or a fragment of a message has one TSN. We need to remember that only DATA chunks use TSNs and only DATA chunks are acknowledged by SACK chunks.

**In SCTP, only data chunks consume TSNs;**
**data chunks are the only chunks that are acknowledged.**

Let us show a simple scenario in Figure 16.20. In this figure a client sends four DATA chunks and receives two DATA chunks from the server. Later, we will discuss, in more detail, the use of flow and error control in SCTP. For the moment, we assume that everything goes well in this scenario. The client uses the verification tag 85, the server 700. The packets sent are described below:

1. The client sends the first packet carrying two DATA chunks with TSNs 7105 and 7106.
2. The client sends the second packet carrying two DATA chunks with TSNs 7107 and 7108.
3. The third packet is from the server. It contains the SACK chunk needed to acknowledge the receipt of DATA chunks from the client. Contrary to TCP, SCTP acknowledges the last in-order TSN received, not the next expected. The third packet also includes the first DATA chunk from the server with TSN 121.
4. After a while, the server sends another packet carrying the last DATA chunk with TSN 122, but it does not include a SACK chunk in the packet because the last DATA chunk received from the client was already acknowledged.
5. Finally, the client sends a packet that contains a SACK chunk acknowledging the receipt of the last two DATA chunks from the server.

**Figure 16.20** *Simple data transfer*




**The acknowledgment in SCTP defines the cumulative TSN, the TSN of the last data chunk received in order.**

### *Multihoming Data Transfer*

We discussed the multihoming capability of SCTP, a feature that distinguishes SCTP from UDP and TCP. Multihoming allows both ends to define multiple IP addresses for

communication. However, only one of these addresses can be defined as the **primary address;** the rest are alternative addresses. The primary address is defined during association establishment. The interesting point is that the primary address of an end is determined by the other end. In other words, a source defines the primary address for a destination.

Data transfer, by default, uses the primary address of the destination. If the primary is not available, one of the alternative addresses is used. The process, however, can always override the primary address and explicitly request that a message be sent to one of the alternative addresses. A process can also explicitly change the primary address of the current association.

A logical question that arises is where to send a SACK. SCTP dictates that a SACK be sent to the address from which the corresponding SCTP packet originated.

### *Multistream Delivery*

One interesting feature of SCTP is the distinction between data transfer and data delivery. SCTP uses TSN numbers to handle data transfer, movement of data chunks between the source and destination. The delivery of the data chunks are controlled by SIs and SSNs. SCTP can support multiple streams, which means that the sender process can define different streams and a message can belong to one of these streams. Each stream is assigned a stream identifier (SI) which uniquely defines that stream. However, SCTP supports two types of data delivery in each stream: **ordered** (default) and **unordered.** In ordered data delivery, data chunks in a stream use stream sequence numbers (SSNs) to define their order in the stream. When the chunks arrive at the destination, SCTP is responsible for message delivery according to the SSN defined in the chunk. This may delay the delivery because some chunks may arrive out of order. In unordered data delivery, the data chunks in a stream have the U flag set, but their SSN field value is ignored. They do not consume SSNs. When an unordered data chunk arrives at the destination SCTP, it delivers the message carrying the chunk to the application without waiting for the other messages. Most of the time, applications use the ordered-delivery service, but occasionally some applications need to send urgent data that must be delivered out of order (recall the urgent data and urgent pointer facility of TCP). In these cases, the application can define the delivery as unordered.

### *Fragmentation*

Another issue in data transfer is **fragmentation.** Although SCTP shares this term with IP, fragmentation in IP and SCTP belong to different levels: the former at the network layer, the latter at the transport layer.

SCTP preserves the boundaries of the message from process to process when creating a DATA chunk from a message if the size of the message (when encapsulated in an IP datagram) does not exceed the MTU of the path. The size of an IP datagram carrying a message can be determined by adding the size of the message, in bytes, to the four overheads: data chunk header, necessary SACK chunks, SCTP general header, and IP header. If the total size exceeds the MTU, the message needs to be fragmented.

Fragmentation at the source SCTP takes place using the following steps:

1. The message is broken into smaller fragments to meet the size requirement.
2. A DATA chunk header is added to each fragment that carries a different TSN. The TSN needs to be in sequence.

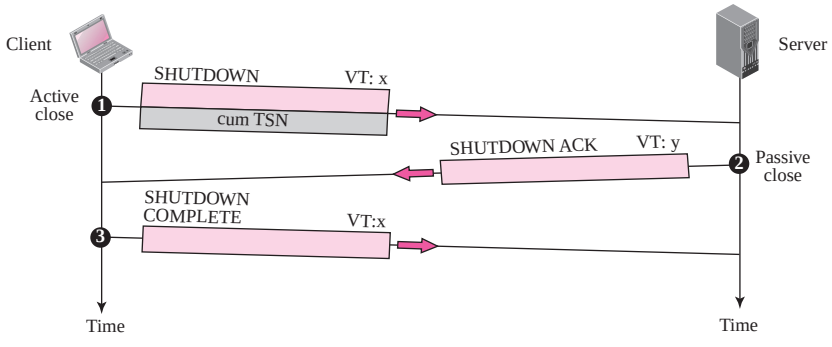
3. All header chunks carry the same stream identifier (SI), the same stream sequence number (SSN), the same payload protocol identifier, and the same U flag.
4. The combination of B and E are assigned as follows:
   a. First fragment: 10
   b. Middle fragments: 00
   c. Last fragment: 01.

The fragments are reassembled at the destination. If a DATA chunk arrives with its B/E bits not equal to 11, it is not fragmented. The receiver knows how to reassemble all chunks with the same SIs and SSNs. The number of fragments is determined by the TSN number of the first and the last fragments.

## Association Termination

In SCTP, like TCP, either of the two parties involved in exchanging data (client or server) can close the connection. However, unlike TCP, SCTP does not allow a "half-closed" association. If one end closes the association, the other end must stop sending new data. If any data are left over in the queue of the recipient of the termination request, they are sent and the association is closed. Association termination uses three packets as shown in Figure 16.21. Note that although the figure shows the case in which termination is initiated by the client, it can also be initiated by the server.

**Figure 16.21** *Association termination*




There can be several scenarios of association termination. We discuss some of them later.

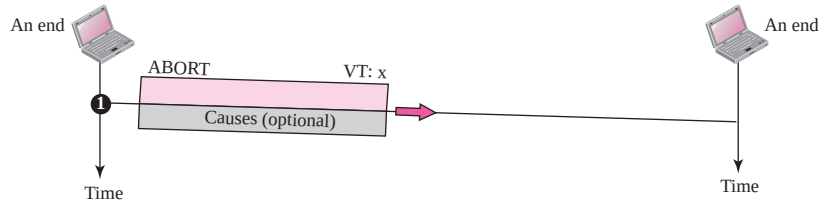
## Association Abortion

The termination of association discussed in the previous section is sometimes referred to as "graceful termination." An association in SCTP can also be aborted. The abortion may be requested by the process at either end or by SCTP. A process may wish to abort the association if there is a problem in the process itself (receiving wrong data from the other end, going into an infinite loop, and so on). The server may wish to abort the association because it has received an INIT chunk with wrong parameters, the

requested resources are not available after receiving the cookie, the operating system needs to shut down, and so on.

The abortion process in SCTP is very simple. Either end can send an ABORT chunk and abort the association as shown in Figure 16.22. No further chunks are needed.

**Figure 16.22** *Association abortion*

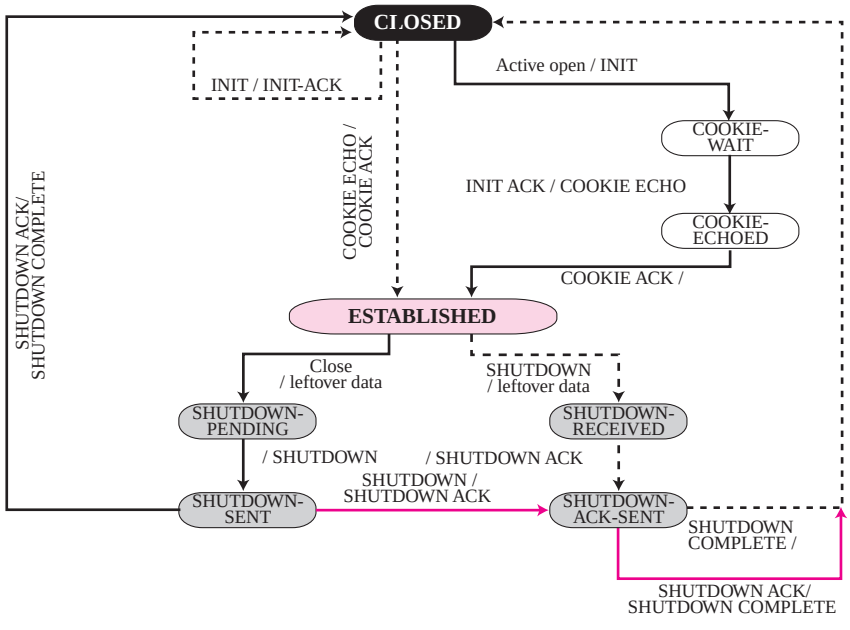



## 16.6 STATE TRANSITION DIAGRAM

To keep track of all the different events happening during association establishment, association termination, and data transfer, the SCTP software, like TCP, is implemented as a finite state machine. Figure 16.23 shows the state transition diagram for both client and server.

The dotted black lines of the figure represent the normal transitions for the server; the solid black lines represent the normal transitions for the client, and the colored lines

**Figure 16.23** *State transition diagram*



represent unusual situations. In special situations, a server can go through a transition shown by a solid line; a client can go through a transition shown by a broken line. Table 16.4 shows the states for SCTP.

**Table 16.4** *States for SCTP*

| *State* | *Description* |
|---|---|
| **CLOSED** | No connection |
| **COOKIE-WAIT** | Waiting for a cookie |
| **COOKIE-ECHOED** | Waiting for cookie acknowledgment |
| **ESTABLISHED** | Connection is established; data are being transferred |
| **SHUTDOWN-PENDING** | Sending data after receiving *close* |
| **SHUTDOWN-SENT** | Waiting for SHUTDOWN acknowledgment |
| **SHUTDOWN-RECEIVED** | Sending data after receiving SHUTDOWN |
| **SHUTDOWN-ACK-SENT** | Waiting for termination completion |

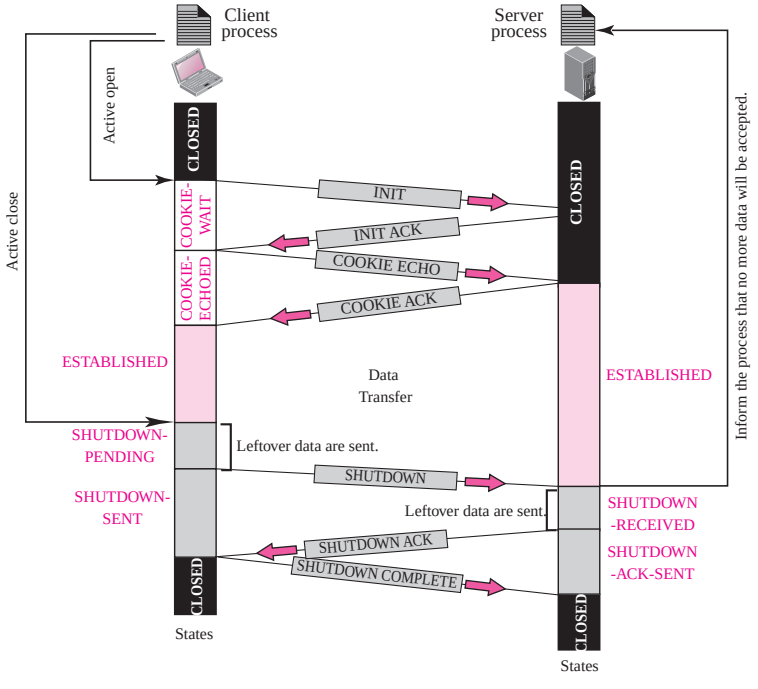
## Scenarios

To understand the SCTP state machines and the transition diagrams, we go through some scenarios in this section.

### *A Common Scenario*

Figure 16.24 shows a typical scenario. This is a routine situation in which the open and close commands come from the client. We have shown the states as we did for the

**Figure 16.24** *A common scenario of states*



corresponding TCP scenario. The figure specifically shows association establishment (the first four packets) and association termination (the last three packets) and the states the client and server go through. Note that the server remains in the **CLOSED** state during association establishment, but the client goes through two states (**COOKIE-WAIT** and **COOKIE-ECHOED**).

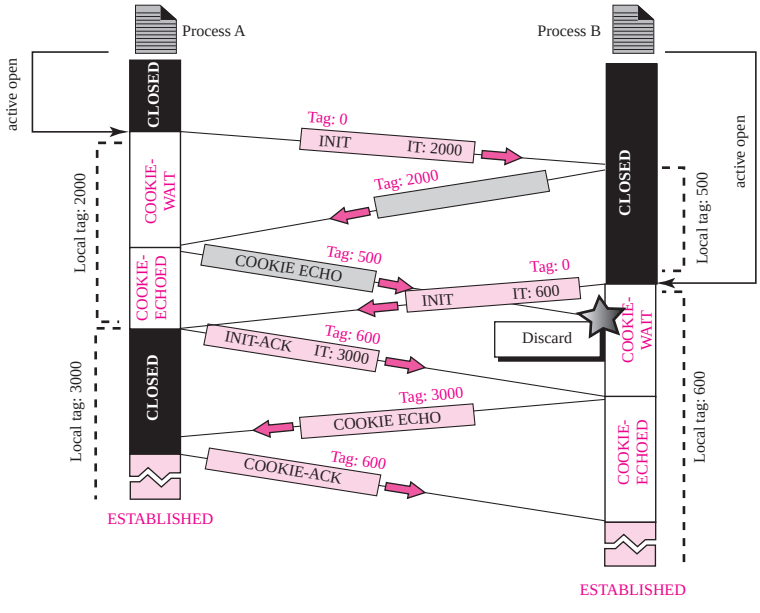
When the client SCTP receives an active close, it goes to the **SHUTDOWN-PENDING** state. It remains in this state until all leftover data is sent. It then sends a SHUTDOWN chunk and goes to **SHUTDOWN-SENT** state. The server, after receiving the SHUTDOWN chunk, informs its process that no more data will be accepted. It then goes to **SHUTDOWN-RECEIVED** state. While in this state, it sends all leftover data to the client and then sends a SHUTDOWN ACK chunk. It then goes to **SHUTDOWN-ACK-SENT** state. After receiving the last chunk, the client sends a SHUTDOWN COMPLETE chunk and closes the association. The server closes the association after receiving the last chunk.

### *Simultaneous Open*

The cookie has created some complexity in association establishment. When an end receives an INIT chunk and sends an INIT-ACK chunk, it is still in the CLOSED state; it does not remember what it has received or sent. This creates a problem if an end process issues an active open, and, before the association is established, the other end process also issues an active open. Figure 16.25 shows a scenario.

The process at site A issues an active open and sends an INIT chunk. Site B receives the INIT chunk and sends an INIT-ACK chunk. Before site B receives a COOKIE-ECHO chunk from site A, the process at site B issues an active open. SCTP

**Figure 16.25** *Simultaneous open*



at site B, not remembering that an association has started, sends an INIT chunk to site A and starts a new association. The two associations collide.

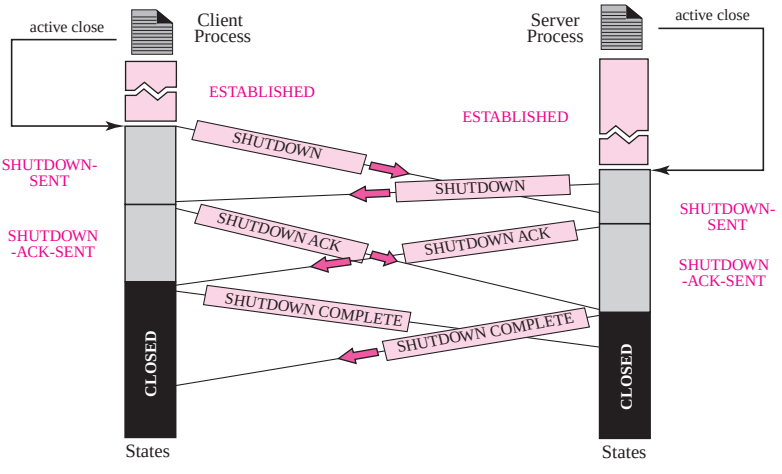
To handle this type of problem, SCTP requires that each site, when sending an INIT or an INIT-ACT chunk, also send an initiation tag. This tag is saved in a variable, say *local tag*. At any time, only one local tag value is held in this variable. Each time a packet arrives with a verification tag that does not match the value of the local tag, it is discarded. SCTP can also start a new association when it is in the middle of association setup and a new INIT chunk arrives.

Our scenario shows the use of these tags. Site A starts with an INIT chunk and initiation tag of 2000; the value of local tag is now 2000. Site B sends an INIT-ACK with an initialization tag of 500; the value of local tag for this site is 500. When site B sends an INIT chunk, the initialization tag and the local tag change to 600. Site B discards the COOKIE-ECHO received because the tag on the packet is 500, which does not match the local tag of 600. The unfinished first association is aborted here. The new association, however, continues with site B as the initiator.

### *Simultaneous Close*

Two end points can close the association simultaneously. This happens when a client's SHUTDOWN chunk reaches the server that has already sent a SHUTDOWN chunk itself. In this case both the client and the server can go to different states to terminate the association as shown in Figure 16.26.

**Figure 16.26** *Simultaneous close*




The figure shows that both the client and the server issue an active close. We assume that none of the SCTPs has leftover data to send. So the **SHUTDOWN PENDING** state is skipped. Both SCTPs go to the **SHUTDOWN-SENT** state after receiving the SHUTDOWN chunks. Both send a SHUTDOWN ACK chunk and go to **SHUTDOWN-ACK-SENT** state. Both remain in this state until they receive the

SHUTDOWN ACK chunk from the other party. They then send SHUTDOWN COMPLETE chunks and go to the closed state. Note that when they receive the last chunks, both SCTPs are already in the **CLOSED** state. If we look at the state transition diagram of Figure 16.23, we see that both the client and server follow the same path after the **ESTABLISHED** state. This is the path the client follows. However, after reaching the SHUTDOWN-SENT state, both turn to the right, instead of the left. When each of them reach the SHUTDOWN-ACK-SENT state, they go downward instead of straight.

### *Other Scenarios*

There are many scenarios that neither space nor time allows us to discuss. We would need to present information about timers as well as the procedures for the receipt of unexpected chunks. We would need additional information on SCTP including appropriate RFCs. We leave these scenarios as exercises or research activities.

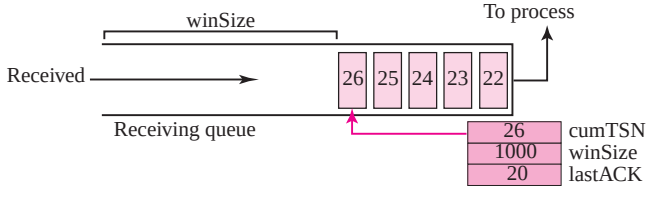
## 16.7 FLOW CONTROL

**Flow control** in SCTP is similar to that in TCP. In TCP, we need to deal with only one unit of data, the byte. In SCTP, we need to handle two units of data, the byte and the chunk. The values of rwnd and cwnd are expressed in bytes; the values of TSN and acknowledgments are expressed in chunks. To show the concept, we make some unrealistic assumptions. We assume that there is never congestion in the network and that the network is error free. In other words, we assume that cwnd is infinite and no packet is lost, delayed, or arrives out of order. We also assume that data transfer is unidirectional. We correct our unrealistic assumptions in later sections. Current SCTP implementations still use a byte-oriented window for flow control. We, however, show a buffer in terms of chunks to make the concept easier to understand.

### Receiver Site

The receiver has one buffer (queue) and three variables. The queue holds the received data chunks that have not yet been read by the process. The first variable holds the last TSN received, *cumTSN*. The second variable holds the available buffer size, *winsize*. The third variable holds the last accumulative acknowledgment, *lastACK*. Figure 16.27 shows the queue and variables at the receiver site.

**Figure 16.27** *Flow control, receiver site*

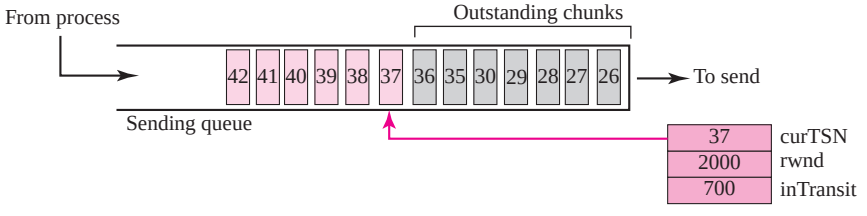


1. When the site receives a data chunk, it stores it at the end of the buffer (queue) and subtracts the size of the chunk from winSize. The TSN number of the chunk is stored in the cumTSN variable.
2. When the process reads a chunk, it removes it from the queue and adds the size of the removed chunk to winSize (recycling).
3. When the receiver decides to send a SACK, it checks the value of lastAck; if it is less than cumTSN, it sends a SACK with a cumulative TSN number equal to the cumTSN. It also includes the value of winSize as the advertised window size. The value of lastACK is then updated to hold the value of cumTSN.

## Sender Site

The sender has one buffer (queue) and three variables: *curTSN*, *rwnd*, and *inTransit*, as shown in Figure 16.28. We assume each chunk is 100 bytes long.

**Figure 16.28** *Flow control, sender site*



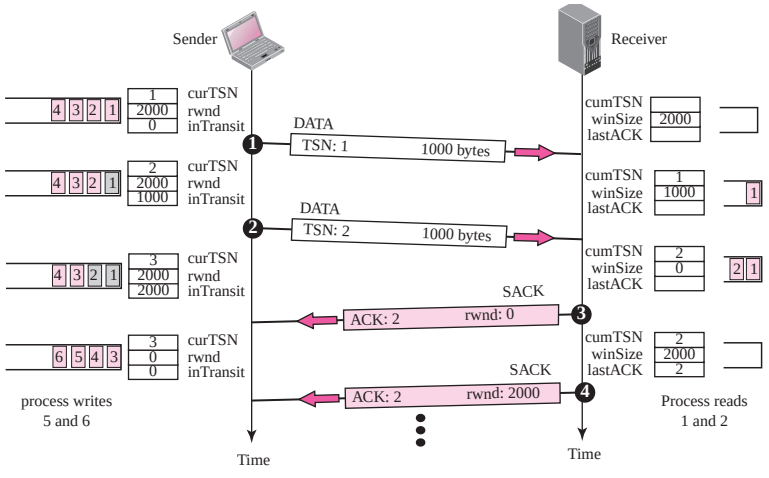

The buffer holds the chunks produced by the process that have either been sent or are ready to be sent. The first variable, curTSN, refers to the next chunk to be sent. All chunks in the queue with a TSN less than this value have been sent, but not acknowledged; they are outstanding. The second variable, rwnd, holds the last value advertised by the receiver (in bytes). The third variable, inTransit, holds the number of bytes in transit, bytes sent but not yet acknowledged. The following is the procedure used by the sender.

1. A chunk pointed to by curTSN can be sent if the size of the data is less than or equal to the quantity (rwnd – inTransit). After sending the chunk, the value of curTSN is incremented by one and now points to the next chunk to be sent. The value of inTransit is incremented by the size of the data in the transmitted chunk.
2. When a SACK is received, the chunks with a TSN less than or equal to the cumulative TSN in the SACK are removed from the queue and discarded. The sender does not have to worry about them any more. The value of inTransit is reduced by the total size of the discarded chunks. The value of rwnd is updated with the value of the advertised window in the SACK.

## A Scenario

Let us give a simple scenario as shown in Figure 16.29. At the start the values of rwnd at the sender site and winSize at the receiver site are both 2000 (advertised during

**Figure 16.29** *Flow control scenario*




association establishment). Originally, there are four messages in the sender queue. The sender sends one data chunk and adds the number of bytes (1000) to the inTransit variable. After awhile, the sender checks the difference between the rwnd and inTransit, which is 1000 bytes, so it can send another data chunk. Now the difference between the two variables is 0 and no more data chunks can be sent. After a while, a SACK arrives that acknowledges data chunks 1 and 2. The two chunks are removed from the queue. The value of inTransit is now 0. The SACK however, advertised a receiver window of value 0, which makes the sender update rwnd to 0. Now the sender is blocked; it cannot send any data chunks (with one exception explained later).

At the receiver site, the queue is empty at the beginning. After the first data chunk is received, there is one message in the queue and the value of cumTSN is 1. The value of winSize is reduced to 1000 because the first message occupies 1000 bytes. After the second data chunk is received, the value of window size is 0 and the cumTSN is 2. Now, as we will see, the receiver is required to send a SACK with cumulative TSN of 2. After the first SACK was sent, the process reads the two messages, which means that there is now room in the queue; the receiver advertises the situation with a SACK to allow the sender to send more data chunks. The remaining events are not shown in the figure.

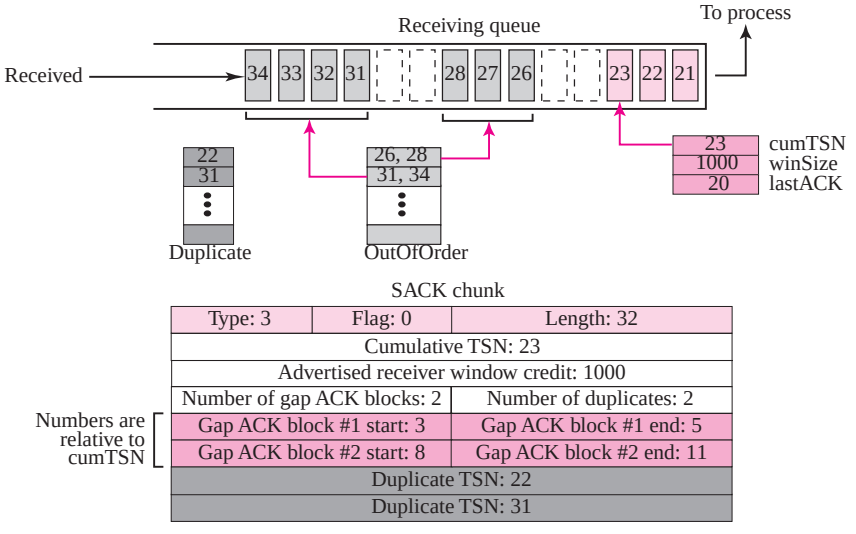
## 16.8 ERROR CONTROL

SCTP, like TCP, is a reliable transport-layer protocol. It uses a SACK chunk to report the state of the receiver buffer to the sender. Each implementation uses a different set of entities and timers for the receiver and sender sites. We use a very simple design to convey the concept to the reader.

## Receiver Site

In our design, the receiver stores all chunks that have arrived in its queue including the out-of-order ones. However, it leaves spaces for any missing chunks. It discards duplicate messages, but keeps track of them for reports to the sender. Figure 16.30 shows a typical design for the receiver site and the state of the receiving queue at a particular point in time.

**Figure 16.30** *Error control, receiver site*



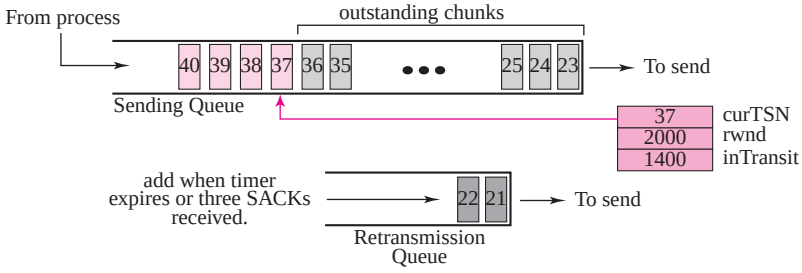

The last acknowledgment sent was for data chunk 20. The available window size is 1000 bytes. Chunks 21 to 23 have been received in order. The first out-of-order block contains chunks 26 to 28. The second out-of-order block contains chunks 31 to 34. A variable holds the value of cumTSN. An array of variables keeps track of the beginning and the end of each block that is out of order. An array of variables holds the duplicate chunks received. Note that there is no need for storing duplicate chunks in the queue, they will be discarded. The figure also shows the SACK chunk that will be sent to report the state of the receiver to the sender. The TSN numbers for out-of-order chunks are relative (offsets) to the cumulative TSN.

## Sender Site

At the sender site, our design demands two buffers (queues): a sending queue and a retransmission queue. We also use three variables: rwnd, inTransit, and curTSN as described in the previous section. Figure 16.31 shows a typical design.

The sending queue holds chunks 23 to 40. The chunks 23 to 36 have already been sent, but not acknowledged; they are outstanding chunks. The curTSN points to the next chunk to be sent (37). We assume that each chunk is 100 bytes, which means that 1400 bytes of data (chunks 23 to 36) are in transit. The sender at this moment has a retransmission queue. When a packet is sent, a retransmission timer starts for that

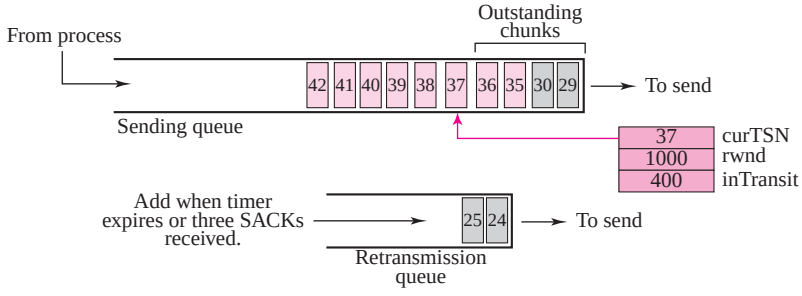
**Figure 16.31** *Error control, sender site*




packet (all data chunks in that packet). Some implementations use one single timer for the entire association, but we continue with our tradition of one timer for each packet for simplification. When the retransmission timer for a packet expires, or three SACKs arrive that declare a packet as missing (fast retransmission was discussed in Chapter 12), the chunks in that packet are moved to the retransmission queue to be resent. These chunks are considered lost, rather than outstanding. The chunks in the retransmission queue have priority. In other words, the next time the sender sends a chunk, it would be chunk 21 from the retransmission queue.

To see how the state of the sender changes, assume that the SACK in Figure 16.30 arrives at the sender site in Figure 16.31. Figure 16.32 shows the new state.

**Figure 16.32** *New state at the sender site after receiving a SACK chunk*




1. All chunks having a TSN equal to or less than the cumTSN in the SACK are removed from the sending or retransmission queue. They are no longer outstanding or marked for retransmission. Chunks 21 and 22 are removed from the retransmission queue and 23 is removed from the sending queue.
2. Our design also removes all chunks from the sending queue that are declared in the gap blocks; some conservative implementations, however, save these chunks until a cumTSN arrives that includes them. This precaution is needed for the rare occasion when the receiver finds some problem with these out-of-order chunks. We ignore these rare occasions. Chunks 26 to 28 and chunks 31 to 34, therefore, are removed from the sending queue.

3. The list of duplicate chunks does not have any effect.
4. The value of rwnd is changed to 1000 as advertised in the SACK chunk.
5. We also assume that the transmission timer for the packet that carried chunks 24 and 25 has expired. These move to the retransmission queue and a new retransmission timer is set according to the exponential backoff rule discussed in Chapter 15.
6. The value of inTransit becomes 400 because only 4 chunks are now in transit. The chunks in the retransmission queue are not counted because they are assumed lost, not in transit.

## Sending Data Chunks

An end can send a data packet whenever there are data chunks in the sending queue with a TSN greater than or equal to curTSN or if there are data chunks in the retransmission queue. The retransmission queue has priority. However, the total size of the data chunk or chunks included in the packet must not exceed the (rwnd – inTransit) value and the total size of the frame must not exceed the MTU size as we discussed in previous sections. If we assume in our previous scenario, that our packet can take 3 chunks (due to the MTU restriction), then chunks 24 and 25 from the retransmission queue and chunk 37, the next chunk ready to be sent in the sending queue, can be sent. Note that the outstanding chunks in the sending queue cannot be sent; they are assumed to be in transit. Note also that any chunk sent from the retransmission queue is also timed for retransmission again. The new timer affects chunks 24, 25, and 37. We need to mention here that some implementations may not allow mixing chunks from the retransmission queue and the sending queue. In this case, only chunks 24 and 25 can be sent in the packet.

### *Retransmission*

To control a lost or discarded chunk, SCTP, like TCP, employs two strategies: using retransmission timers and receiving four SACKs with the same missing chunks.

**Retransmission Timer** SCTP uses a retransmission timer, which handles the retransmission time, the waiting time for an acknowledgment of a segment. The procedures for calculating RTO and RTT in SCTP are the same as we described for TCP. SCTP uses a measured RTT ($RTT_M$), a smoothed RTT ($RTT_S$), and an RTT deviation ($RTT_D$) to calculate the RTO. SCTP also uses Karn's algorithm to avoid acknowledgment ambiguity. Note that if a host is using more than one IP address (multihoming), separate RTOs must be calculated and kept for each path.

**Four Missing Reports** Whenever a sender receives four SACKs whose gap ack information indicate one or more specific data chunks are missing, the sender needs to consider those chunks as lost and immediately move them to the retransmission queue. This behavior is analogous to "fast retransmission" in TCP, discussed in Chapter 15.

## Generating SACK Chunks

Another issue in error control is the generation of SACK chunks. The rules for generating SCTP SACK chunks are similar to the rules used for acknowledgment with the TCP ACK flag. We summarize the rules as listed below.

1. When an end sends a DATA chunk to the other end, it must include a SACK chunk advertising the receipt of unacknowledged DATA chunks.
2. When an end receives a packet containing data, but has no data to send, it needs to acknowledge the receipt of the packet within a specified time (usually 500 ms).
3. An end must send at least one SACK for every other packet it receives. This rule overrides the second rule.
4. When a packet arrives with out-of-order data chunks, the receiver needs to immediately send a SACK chunk reporting the situation to the sender.
5. When an end receives a packet with duplicate DATA chunks and no new DATA chunks, the duplicate data chunks must be reported immediately with a SACK chunk.

## 16.9 CONGESTION CONTROL

SCTP, like TCP, is a transport layer protocol with packets subject to congestion in the network. The SCTP designers have used the same strategies we described for congestion control in Chapter 15 for TCP. SCTP has slow start (exponential increase), congestion avoidance (additive increase), and congestion detection (multiplicative decrease) phases. Like TCP, SCTP also uses fast retransmission and fast recovery.

### Congestion Control and Multihoming

Congestion control in SCTP is more complicated since the host may have more than one IP address. In this case, there can be more than one path for the data in the network. Each of these paths may encounter different levels of congestion. This implies that the site needs to have different values of cwnd for each IP address.

### Explicit Congestion Notification

Explicit congestion notification (ECN), as defined for other wide area networks, is a process that enables a receiver to explicitly inform the sender of any congestion experienced in the network. If a receiver encounters many delayed or lost packets, it is an indication of probable congestion. SCTP can use an ECN option in the INIT and INIT ACK chunks to allow both ends to negotiate the use of ECN. If both parties agree, the receiver can inform the sender of congestion by sending an ECNE (explicit congestion notification echo) chunk with each packet until it receives a CWR (congestion window reduce) chunk to show that the sender has reduced its cwnd. We have not discussed these two chunks because they are not yet part of the standard and because the discussion of explicit congestion notification is beyond the scope of this book.

## 16.10 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

We highly recommend [Ste & Xie 01], which is totally devoted to the SCTP protocol.

### RFCs

SCTP is discussed in many RFCs including RFC 4820, RFC 4895, RFC 4960, RFC 5043, RFC 5061, and RFC 5062.

## 16.11 KEY TERMS

ABORT chunk
association
association establishment
byte-oriented
CLOSED state
cookie
COOKIE ACK chunk
COOKIE ECHO chunk
COOKIE-ECHOED state
COOKIE-WAIT state
DATA chunk
ERROR chunk
ESTABLISHED state
fragmentation
HEARTBEAT ACK chunk
HEARTBEAT chunk
INIT-ACK chunk
INIT chunk
initial TSN
initiation tag
message-oriented
multihoming service
multistream service
ordered delivery
primary address
SACK chunk
SHUTDOWN ACK chunk
SHUTDOWN chunk
SHUTDOWN COMPLETE chunk
SHUTDOWN-ACT-SENT state
SHUTDOWN-PENDING state
SHUTDOWN-RECEIVED state
SHUTDOWN-SENT state
stream identifier (SI)
stream sequence number (SSN)
transmission sequence number (TSN)
unordered delivery
verification tag

## 16.12 SUMMARY

- SCTP is a message-oriented, reliable protocol that combines the good features of UDP and TCP. SCTP provides additional services not provided by UDP or TCP, such as multiple-stream and multihoming services. SCTP is a connection-oriented protocol. An SCTP connection is called an association. SCTP provides flow control, error control, and congestion control.
- SCTP uses the term *packet* to define a transportation unit. In SCTP, control information and data information are carried in separate chunks.
- To distinguish between different streams, SCTP uses the sequence identifier (SI). To distinguish between different data chunks belonging to the same stream, SCTP uses the stream sequence number (SSN). Data chunks are identified by three identifiers: TSN, SI, and SSN.

- SCTP acknowledgment numbers are used only to acknowledge data chunks; control chunks are acknowledged, if needed, by another control chunk.
- SCTP has states within a transition diagram. The states defined for SCTP are **CLOSED**, **COOKIE-WAIT**, **COOKIE-ECHOED**, **ESTABLISHED**, **SHUTDOWN-PENDING**, **SHUTDOWN-SENT**, **SHUTDOWN-RECEIVED**, and **SHUTDOWN-ACK-SENT**.
- A DATA chunk cannot carry data belonging to more than one message, but a message can be split into several chunks (fragmentation).
- An SCTP association is normally established using four packets (four-way handshaking). An association is normally terminated using three packets (three-way handshaking). An SCTP association uses a cookie to prevent blind flooding attacks and a verification tag to avoid insertion attacks.
- The SCTP acknowledgment SACK reports the cumulative TSN, the TSN of the last data chunk received in order, and selective TSN that have been received.

## 16.13 PRACTICE SET

### Exercises

1. A packet is carrying two DATA chunks, each containing 22 bytes of user data. What is the size of each DATA chunk? What is the total size of the packet?
2. A SACK chunk reports the receipt of three out-of-order data chunks and five duplicate data chunks. What is the total size of the chunk in bytes?
3. A packet is carrying a COOKIE ECHO message and a DATA chunk. If the size of the cookie is 200 bytes and that of the user data is 20 bytes, what is the size of the packet?
4. A packet is carrying a COOKIE ACK message and a DATA chunk. If the user data is 20 bytes, what is the size of the packet?
5. Four DATA chunks have arrived carrying the following information:

| | | | |
|---|---|---|---|
| TSN:27 | SI:2 | SSN:14 | BE:00 |
| TSN:33 | SI:2 | SSN:15 | BE:11 |
| TSN:26 | SI:2 | SSN:14 | BE:00 |
| TSN:24 | SI:2 | SSN:14 | BE:00 |
| TSN:21 | SI:2 | SSN:14 | BE:10 |

   **a.** Which data chunk is a fragment?
   **b.** Which data chunk is the first fragment?
   **c.** Which data chunk is the last fragment?
   **d.** How many middle fragments are missing?

6. The value of the cumulative TSN in a SACK is 23. The value of the previous cumulative TSN in the SACK was 29. What is the problem?
7. An SCTP association is in the **ESTABLISHED** state. It receives a SHUTDOWN chunk. If the host does not have any outstanding or pending data, what does it need to do?
8. An SCTP association is in the **COOKIE-WAIT** state. It receives an INIT chunk; what does it need to do?
9. The following is a dump of a DATA chunk in hexadecimal format.

```
00000015 00000005 0003000A 00000000 48656C6C 6F000000
```

   a. Is this an ordered or unordered chunk?
   b. Is this the first, the last, the middle, or the only fragment?
   c. How many bytes of padding are carried by the chunk?
   d. What is the TSN?
   e. What is the SI?
   f. What is the SSN?
   g. What is the message?

10. The following is a dump of an SCTP general header in hexadecimal format.

```
04320017 00000001 00000000
```

   a. What is the source port number?
   b. What is the destination port number?
   c. What is the value of the verification tag?
   d. What is the value of the checksum?

11. The state of a receiver is as follows:
   a. The receiving queue has chunks 1 to 8, 11 to 14, and 16 to 20.
   b. There are 1800 bytes of space in the queue.
   c. The value of lastAck is 4.
   d. No duplicate chunk has been received.
   e. The value of cumTSN is 5.

   Show the contents of the receiving queue and the variables.
12. Show the contents of the SACK message sent by the receiver in Exercise 11.
13. The state of a sender is as follows:
   a. The sending queue has chunks 18 to 23.
   b. The value of curTSN is 20.
   c. The value of the window size is 2000 bytes.
   d. The value of inTransit is 200.

   If each data chunk contains 100 bytes of data, how many DATA chunks can be sent now? What is the next data chunk to be sent?
14. An SCTP client opens an association using an initial tag of 806, an initial TSN of 14534, and a window size of 20000. The server responds with an initial tag of

2000, an initial TSN of 670, and a window size of 14000. Show the contents of all four packets exchanged during association establishment. Ignore the value of the cookie.

**15.** If the client in the previous exercise sends 7600 data chunks and the server sends 570 data chunks, show the contents of the three packets exchanged during association termination.

## Research Activities

**16.** We have defined only a few scenarios related to the transition diagram. Show these other scenarios including, but not limited to, the following cases:

- **a.** The loss, delay, or duplication of any of the four packets during association establishment.
- **b.** The loss, delay, or duplication of any of the three packets during association termination.

**17.** What happens if a SACK chunk is delayed or lost?

**18.** Find the steps a server takes to validate a cookie.

**19.** We discussed two timers in TCP: persistence and keepalive. Find out the functionality of these timers in SCTP.

**20.** Find out more about ECN in SCTP. Find the format of these two chunks.

**21.** Find out more about the parameters used in some SCTP control chunks.

**22.** Some application programs, such as FTP, need more than one connection when using TCP. Find how the multistream service of SCTP can help these applications establish only one association with several streams.

# PART 4

# Application Layer

CHAPTER

# 17 Introduction to the Application Layer

This chapter is an introduction to the application layer. In the next eight chapters we introduce common client-server applications used in the Internet. In this chapter, we give a general picture of how a client-server program is designed and give some simple codes of their implementation. The area of network programming is a very vast and complicated one; it cannot be covered in one chapter. We need to give a bird's-eye view of this discipline to make the contents of the next eight chapters easier to understand.

## OBJECTIVES

*The chapter has several objectives:*

- To introduce client-server paradigm.
- To introduce socket interfaces and list some common functions in this interface.
- To discuss client-server communication using connectionless iterative service offered by UDP.
- To discuss client-server communication using connection-oriented concurrent service offered by TCP.
- To give an example of a client program using UDP.
- To give an example of a server program using UDP.
- To give an example of a client program using TCP.
- To give an example of server program using TCP.
- To briefly discuss the peer-to-peer paradigm and its application.

## 17.1 CLIENT-SERVER PARADIGM

The purpose of a network, or an internetwork, is to provide services to users: A user at a local site wants to receive a service from a computer at a remote site. One way to achieve this purpose is to run two programs. A local computer runs a program to request a service from a remote computer; the remote computer runs a program to give service to the requesting program. This means that two computers, connected by an internet, must each run a program, one to provide a service and one to request a service.

At first glance, it looks simple to enable communication between two application programs, one running at the local site, the other running at the remote site. But many questions arise when we want to implement the approach. Some of the questions that we may ask are:

1. Should both application programs be able to request services and provide services or should the application programs just do one or the other? One solution is to have an application program, called the *client*, running on the local machine, request a service from another application program, called the *server*, running on the remote machine. In other words, the tasks of requesting a service and providing a service are separate from each other. An application program is either a requester (a client), or a provider (a server). In other words, application programs come in pairs, client and server, both having the same name.
2. Should a server provide services only to one specific client or should the server be able to provide services to any client that requests the type of service it provides? The most common solution is a server providing a service for any client that needs that type of service, not a particular one. In other words, the server-client relationship is one-to-many.
3. Should a computer run only one program (client or server)? The solution is that any computer connected to the Internet should be able to run any client program if the appropriate software is available. The server programs need to be run on a computer that can be continuously running as we will see later.
4. When should an application program be running? All of the time or just when there is a need for the service? Generally, a client program, which requests a service, should run only when it is needed. The server program, which provides a service, should run all the time because it does not know when its service will be needed.
5. Should there be only one universal application program that can provide any type of service a user wants? Or should there be one application program for each type of service? In TCP/IP, services needed frequently and by many users have specific client-server application programs. For example, we have separate client-server application programs that allow users to access files, send e-mail, and so on. For

services that are more customized, we should have one generic application program that allows users to access the services available on a remote computer. For example, we should have a client-server application program that allows the user to log onto a remote computer and then use the services provided by that computer.

## Server

A *server* is a program running on the remote machine providing service to the clients. When it starts, it opens the door for incoming requests from clients, but it never initiates a service until it is requested to do so.

A server program is an *infinite* program. When it starts, it runs infinitely unless a problem arises. It waits for incoming requests from clients. When a request arrives, it responds to the request, either iteratively or concurrently as we will see shortly.

## Client

A *client* is a program running on the local machine requesting service from a server. A client program is *finite*, which means it is started by the user (or another application program) and terminates when the service is complete. Normally, a client opens the communication channel using the IP address of the remote host and the well-known port address of the specific server program running on that machine. After a channel of communication is opened, the client sends its request and receives a response. Although the request-response part may be repeated several times, the whole process is finite and eventually comes to an end.

## Concurrency

Both clients and servers can run in concurrent mode.

### *Concurrency in Clients*

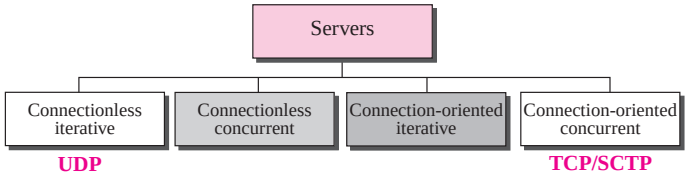
Clients can be run on a machine either iteratively or concurrently. Running clients *iteratively* means running them one by one; one client must start, run, and terminate before the machine can start another client. Most computers today, however, allow *concurrent* clients; that is, two or more clients can run at the same time.

### *Concurrency in Servers*

An *iterative* server can process only one request at a time; it receives a request, processes it, and sends the response to the requestor before it handles another request. A concurrent server, on the other hand, can process many requests at the same time and thus can share its time between many requests.

The servers use either UDP, a connectionless transport layer protocol, or TCP/SCTP, a connection-oriented transport layer protocol. Server operation, therefore, depends on two factors: the transport layer protocol and the service method. Theoretically we can have four types of servers: connectionless iterative, connectionless concurrent, connection-oriented iterative, and connection-oriented concurrent (see Figure 17.1).

**Figure 17.1** *Server types*

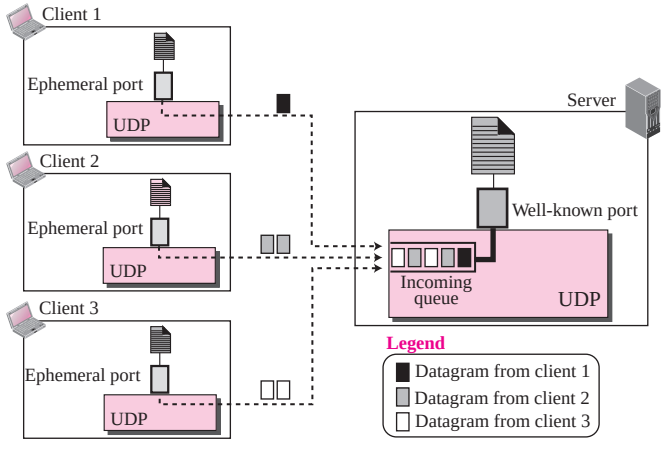



### *Connectionless Iterative Server*

The servers that use UDP are normally iterative, which, as we have said, means that the server processes one request at a time. A server gets the request received in a datagram from UDP, processes the request, and gives the response to UDP to send to the client. The server pays no attention to the other datagrams. These datagrams are stored in a queue, waiting for service. They could all be from one client or from many clients. In either case they are processed one by one in order of arrival.

The server uses one single port for this purpose, the well-known port. All the datagrams arriving at this port wait in line to be served, as is shown in Figure 17.2.

**Figure 17.2** *Connectionless iterative server*



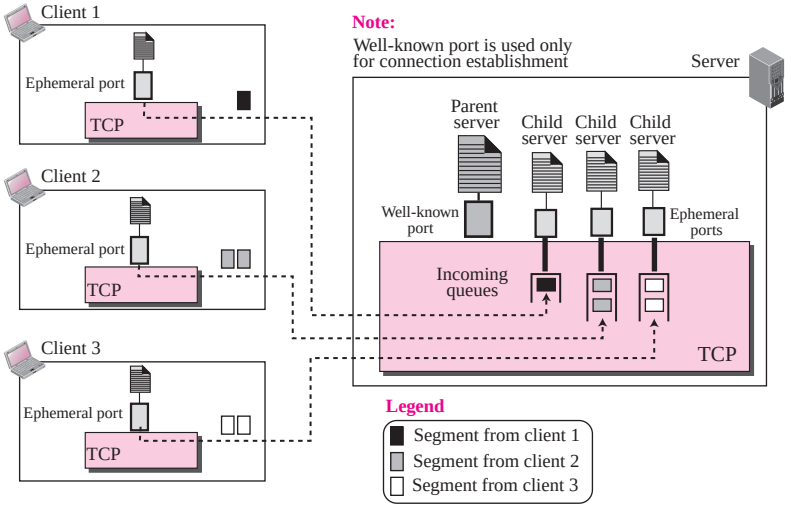

### *Connection-Oriented Concurrent Server*

The servers that use TCP (or SCTP) are normally concurrent. This means that the server can serve many clients at the same time. Communication is connection-oriented, which means that a request is a stream of bytes that can arrive in several segments and the response can occupy several segments. A connection is established between the server and each client, and the connection remains open until the entire stream is processed and the connection is terminated.

This type of server cannot use only one port because each connection needs a port and many connections may be open at the same time. Many ports are needed, but a server can use only one well-known port. The solution is to have one well-known port and many ephemeral ports. The server accepts connection requests at the well-known port. A client can make its initial approach to this port to make the connection. After the connection is made, the server assigns a temporary port to this connection to free the well-known port. Data transfer can now take place between these two temporary ports, one at the client site and the other at the server site. The well-known port is now free for another client to make the connection. To serve several clients at the same time, a server creates child processes, which are copies of the original process (parent process).

The server must also have one queue for each connection. The segments come from the client, are stored in the appropriate queue, and will be served concurrently by the server. See Figure 17.3 for this configuration.

**Figure 17.3** *Connection-oriented concurrent server*




## Socket Interfaces

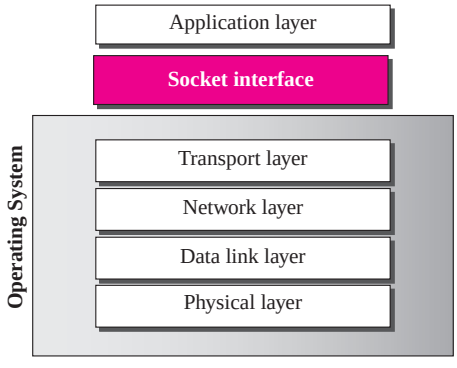
How can a client process communicate with a server process? A computer program is a set of predefined instructions that tells the computer what to do. A computer program has a set of instructions for mathematical operations, another set of instructions for string manipulation, still another set of instructions for input/output access. If we need a program to be able to communicate with another program running on another machine, we need a new set of instructions to tell the transport layer to open the connection, send data to and receive data from the other end, and close the connection. A set of instructions of this kind is normally referred to as an **interface.**

**An interface is a set of instructions designed for interaction between two entities.**

Several interfaces have been designed for communication. Three among them are common: **socket interface**, **transport layer interface** (**TLI**), and **STREAM.** Although a network programmer needs to be familiar with all of these interfaces, we briefly discuss only the socket interface in this chapter to give the general idea of network communication at the application layer.

Socket interface started in early 1980s at the University of Berkeley as part of a UNIX environment. To better understand the concept of socket interface, we need to consider the relationship between the underlying operating system, such as UNIX or Windows, and the TCP/IP protocol suite. The issue that we have ignored so far. Figure 17.4 shows a conceptual relation between the operating system and the suite.

**Figure 17.4** *Relation between the operating system and the TCP/IP suite*




The socket interface, as a set of instructions, is located between the operating system and the application programs. To access the services provided by the TCP/IP protocol suite, an application needs to use the instructions defined in the socket interface.

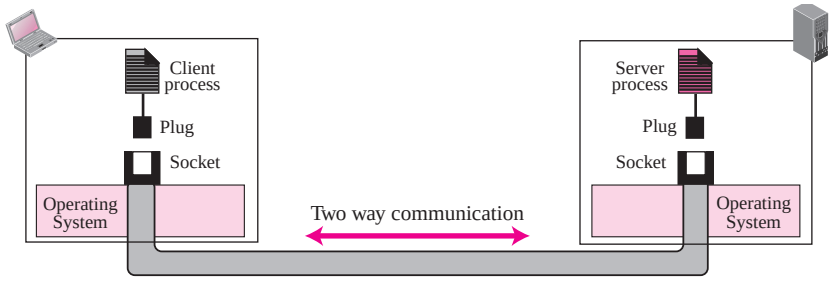
### Example 17.1

Most of the programming languages have a *file interface*, a set of instructions that allow the programmer to open a file, read from the file, write to the file, perform other operations on the file, and finally close the file. When a program needs to open the file, it uses the name of the file as it is known to the operation system. When the file is opened, the operating system returns a reference to the file (an integer or pointer) that can be used for other instructions, such as read and write.

### *Socket*

A **socket** is a software abstract simulating a hardware socket we see in our daily life. To use the communication channel, an application program (client or server) needs to request the operating system to create a socket. The application program then can *plug* into the socket to send and receive data. For data communication to occur, a pair of sockets, each at one end of communication, is needed. Figure 17.5 simulates this abstraction using the socket and plug that we use in our daily life (for a telephone, for example); in the Internet a socket is a software data structure as we discuss shortly.

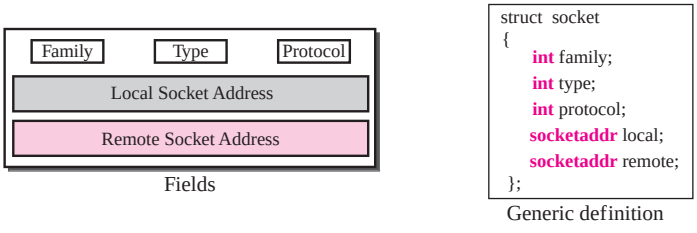
**Figure 17.5** *Concept of sockets*




### *Data Structure*

The format of data structure to define a socket depends on the underlying language used by the processes. For the rest of this chapter, we assume that the processes are written in C language. In C language, a socket is defined as a five-field structure (struct or record) as shown in Figure 17.6.

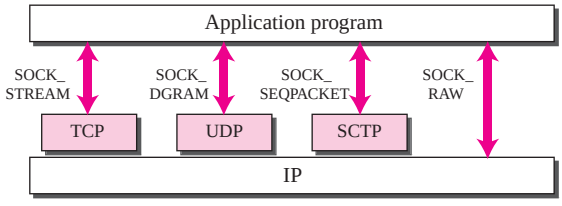
**Figure 17.6** *Socket data structure*




Note that the programmer should not redefine this structure; it is already defined. The programmer needs only to use the header file that includes this definition (discussed later). Let us briefly define the field uses in this structure:

- **Family.** This field defines the protocol group: IPv4, IPv6, UNIX domain protocols, and so on. The family type we use in TCP/IP is defined by the constant IF_INET for IPv4 protocol and IF_INET6 for IPv6 protocol.
- **Type.** This field defines four types of sockets: SOCK_STREAM (for TCP), SOCK_DGRAM (for UDP), SOCK_SEQPACKET (for SCTP), and SOCK_RAW (for applications that directly use the services of IP. They are shown in Figure 17.7.
- **Protocol.** This field defines the protocol that uses the interface. It is set to 0 for TCP/IP protocol suite.
- **Local socket address.** This field defines the local socket address. A socket address, as discussed in Chapter 13, is a combination of an IP address and a port number.
- **Remote socket address.** This field defines the remote socket address.

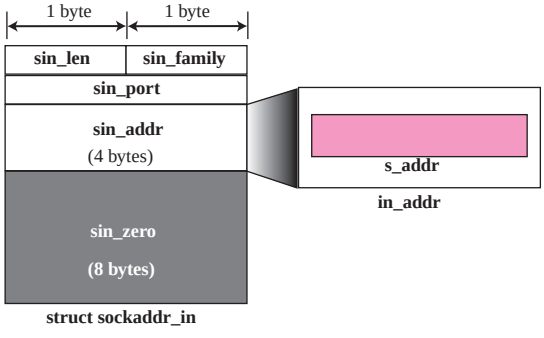
**Figure 17.7** *Socket types*




### *Structure of a Socket Address*

Before we can use a socket, we need to understand the structure of a socket address, a combination of IP address and port number. Although several types of socket addresses have been defined in network programming, we define only the one used for IPv4. In this version a socket address is a complex data structure (struct in C) as shown in Figure 17.8.

**Figure 17.8** *IPv4 socket address*




Note that the struct ***sockaddr_in*** has five fields, but the ***sin_addr*** field is itself a struct of type **in_addr** with a one single field ***s_addr.*** We first define the structure ***in_addr*** as shown below:

```
struct   in_addr
{
    in_addr_t          s_addr;          // A 32-bit IPv4 address
}
```

We now define the structure of **sockaddr_in**:

```
struct   sockaddr_in
{
    uint8_t          sin_len;        // length of structure (16 bytes)
    sa_family_t      sin_family;     // set to AF_INET
    in_port_t        sin_port;       // A 16-bit port number
    struct in_addr   sin_addr;       // A 32-bit IPv4 address
    char             sin_zero[8];    // unused
}
```

### *Functions*

The interaction between a process and the operating system is done through a list of predefined functions. In this section, we introduce these functions; in later sections, we show how they are combined to create processes.

#### ❑ The *socket* Function

The operating system defines the socket structure shown in Figure 17.6. The operating system, however, does not create a socket until instructed by the process. The process needs to use the *socket* function call to create a socket. The prototype for this function is shown below:

```
int socket (int family, int type, int protocol);
```

A call to this function creates a socket, but only three fields in the socket structure (family, type, and protocol) are filled. If the call is successful, the function returns a unique socket descriptor *sockfd* (a non-negative integer) that can be used to refer to the socket in other calls; if the call is not successful, the operating system returns –1.

#### ❑ The *bind* Function

The socket function fills the fields in the socket partially. To bind the socket to the local computer and local port, the *bind* function needs to be called. The bind function, fills the value for the local socket address (local IP address and local port number). It returns –1 if the binding fails. The prototype is shown below:

```
int bind (int sockfd, const struct sockaddress* localAddress, socklen_t addrLen);
```

In this prototype, *sockfd* is the value of the socket descriptor returned from the socket function call, *localAddress* is a pointer to a socket address that needs to have been defined (by the system or the programmer), and the *addrLen* is the length of the socket address. We will see later when the bind function is and is not used.

### ❑ The *connect* Function

The connect function is used to add the remote socket address to the socket structure. It returns –1 if the connection fails.The prototype is given below:

```
int connect (int sockfd, const struct sockaddress* remoteAddress, socklen_t addrLen);
```

The argument is the same, except that the second and third argument defines the remote address instead of the local one.

### ❑ The *listen* Function

The listen function is called only by the TCP server. After TCP has created and bound a socket, it must inform the operating system that a socket is ready for receiving client requests. This is done by calling the *listen* function. The backlog is the maximum number of connection requests. The function returns –1 if it fails. The following shows the prototype:

```
int listen (int sockfd, int backlog);
```

### ❑ The *accept* Function

The accept function is used by a server to inform TCP that it is ready to receive connections from clients. This function returns –1 if it fails. Its prototype is shown below:

```
int accept (int sockfd, const struct sockaddress* clientAddr, socklen_t* addrLen);
```

The last two arguments are pointers to address and to length. The accept function is a blocking function that, when called, blocks itself until a connection is made by a client. The accept function then gets the client socket address and the address length and passes it to the server process to be used to access the client. Note several points:

**a.** The call to accept function makes the process check if there is any client connection request in the waiting buffer. If not, the accept makes the process to sleep. The process wakes up when the queue has at least one request.

**b.** After a successful call to the accept, a new socket is created and the communication is established between the client socket and the new socket of the server.

**c.** The address received from the *accept* function fills the remote socket address in the new socket.

**d.** The address of the client is returned via a pointer. If the programmer does not need this address, it can be replaced by NULL.

**e.** The length of address to be returned is passed to the function and also returned via a pointer. If this length is not needed, it can be replaced by NULL.

### ❑ The *fork* function

The *fork* function is used by a process to duplicate a process. The process that calls the *fork* function is referred to as the parent process; the process that is created, the

duplicate, is called the child process. The prototype is shown below:

```
pid_t fork (fork);
```

It is interesting to know that the fork process is called once, but it returns twice. In the parent process, the return value is a positive integer (the process ID of the parent process that called it). In the child process, the return value is 0. If there is an error, the fork function returns –1. After the fork, two processes are running concurrently; the CPU gives running time to each process alternately.

- **The *send* and *recv* Functions**

The *send* function is used by a process to send data to another process running on a remote machine. The *recv* function is used by a process to receive data from another process running on a remote machine.These functions assume that there is already an open connection between two machines; therefore, it can only be used by TCP (or SCTP). These functions returns the number of bytes send or receive.

```
int send (int sockfd, const void* sendbuf, int nbytes, int flags);

int recv (int sockfd, void* recvbuf, int nbytes, int flags);
```

Here *sockfd* is the socket descriptor; *sendbuf* is a pointer to the buffer where data to be sent have been stored; *recvbuf* is a pointer to the buffer where the data received is to be stored. *nbytes* is the size of data to be sent or received. This function returns the number of actual bytes sent or received if successful and –1 if there is an error.

- **The *sendto* and *recvfrom* Functions**

The *sendto* function is used by a process to send data to a remote process using the services of UDP. The *recvfrom* function is used by a process to receive data from a remote process using the services of UDP. Since UDP is a connectionless protocol, one of the arguments defines the remote socket address (destination or source).

```
int sendto (int sockfd, const void* buffer, int nbytes, int flags
            struct sockaddr* destinationAddress, socklen_t addrLen);

int recvfrom (int sockfd, void* buffer, int nbytes, int flags
              struct sockaddr* sourceAddress, socklen_t* addrLen);
```

Here *sockfd* is the socket descriptor, *buffer* is a pointer to the buffer where data to be sent or to be received is stored, and *buflen* is the length of the buffer. Although, the value of the flag can be nonzero, we let it be 0 for our simple programs in this chapter. These functions return the number of bytes sent or received if successful and –1 if there is an error.

### ❑ The *close* Function

The close function is used by a process to close a socket.

```
int close (int sockfd);
```

The *sockfd* is not valid after calling this function. The socket returns an integer, 0 for success and –1 for error.

### ❑ Byte Ordering Functions

Information in a computer is stored in *host byte order*, which can be *little-endian*, in which the little-end byte (least significant byte) is stored in the starting address, or *big-endian*, in which the big-end byte (most significant byte) is stored in the starting address. Network programming needs data and other pieces of information to be in *network byte order*, which is big-endian. Since when we write programs, we are not sure, how the information such as IP addresses and port number are stored in the computer, we need to change them to network byte order. Two functions are designed for this purpose: ***htons*** (host to network short), which changes a short (16-bit) value to a network byte order, and ***htonl*** (host to network long), which does the same for a long (32-bit) value. There are also two functions that do exactly the opposite: ***ntohs*** and ***ntohl.*** The prototype of these functions are shown below:

```
uint16_t htons (uint16_t shortValue);
uint32_t htonl (uint32_t longValue);
uint16_t ntohs (uint16_t shortValue);
uint32_t ntohl (uint32_t longValue);
```

### ❑ Memory Management Functions

Finally, we need some functions to manage values stored in the memory. We introduce three common memory functions here, although we do not use all of them in this chapter.

```
void* memset (void* destination, int chr, size_t len);
void* memcpy (void* destination, const void* source, size_t nbytes);
int memcmp (const void* ptr1, const void* ptr2, size_t nbytes);
```

The first function, *memset* (memory set) is used to set (store) a specified number of bytes (value of *len*) in the memory defined by the destination pointer (starting address). The second function, *memcpy* (memory copy) is used to copy a specified number of bytes (value of *nbytes*) from part of a memory (source) to another part of memory (destination). The third function, *memcmp*

(memory compare), is used to compare two sets of bytes (nbytes) starting from ptr1 and ptr2. The result is 0 if two sets are equal, less than zero if the first set is smaller than the second, and greater than zero if the first set is larger than the second. The comparison is based on comparing strings of bytes in the C language.

❑ **Address Conversion Functions**

We normally like to work with 32-bit IP address in dotted decimal format. When we want to store the address in a socket, however, we need to change it to a number. Two functions are used to convert an address from a presentation to a number and vice versa: ***inet_pton*** (presentation to number) and ***inet_ntop*** (number to presentation). The constant use for family value is AF_INET for our purpose. Their prototypes are shown below:

```
int inet_pton (int family, const char* stringAddr, void* numericAddr);
char* inet_ntop (int family, const void* numericAddr, char* stringAddr, int len);
```

### *Header Files*

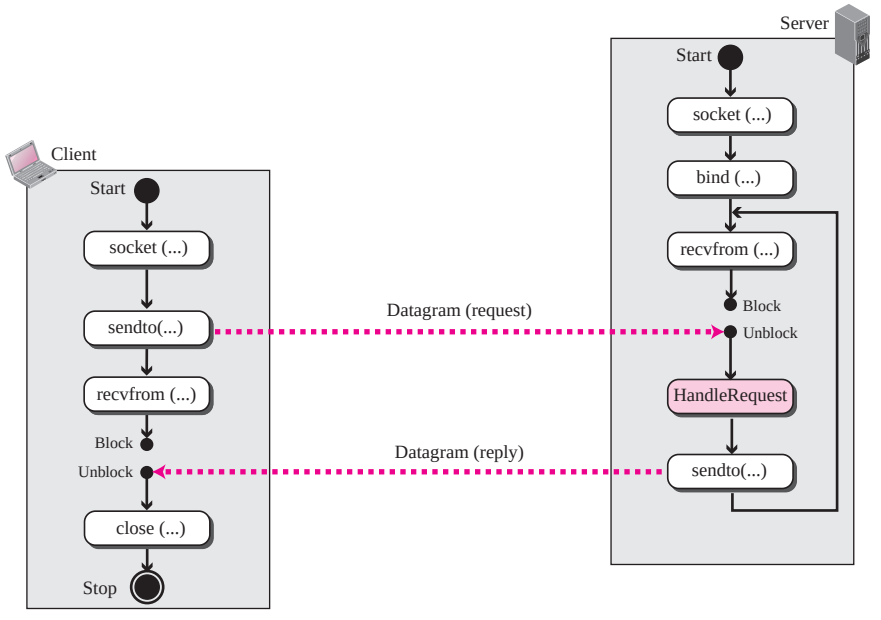
To use previously described functions, we need a set of header files. We define this header file in a separate file, which we name "headerFiles.h". We include this file in our programs to avoid including long lists of header files. Not all of these header files may be needed in all programs, but it is recommended to include all of them in case.

```
// "headerFiles.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <errno.h>
#include <signal.h>
#include <unistd.h>
#include <string.h>
#include <arpa/innet.h>
#include <sys/wait.h>
```

## Communication Using UDP

Figure 17.9 shows a simplified flow diagram for this type of communication.

**Figure 17.9** *Connectionless iterative communication using UDP*




### *Server Process*

The server process starts first. The server process calls the *socket* function to create a socket. It then calls the *bind* function to bind the socket to its well-known port and the IP address of the computer on which the server process is running. The server then calls the *recvfrom* function, which blocks until a datagram arrives. When the datagram arrives, the *recvfrom* function unblocks, extracts the client socket address and address length from the received datagram, and returns them to the process. The process saves these two pieces of information and calls a procedure (function) to handle the request. When the result is ready, the server process calls the *sendto* function and uses the saved information to send the result to the client that requested it. The server uses an infinite loop to respond to the requests coming from the same client or different clients.

### *Client Process*

The client process is simpler. The client calls the *socket* function to create a socket. It then calls the *sendto* function and pass the socket address of the server and the location of the buffer from which UDP can get the data to make the datagram. The client then calls a *recvfrom* function call that blocks until the reply arrives from the server. When the reply arrives, UDP delivers the data to the client process, which make the recv function to unblock and deliver the data received to the client process. Note that we assume that the client message is so small that it fits into one single datagram. If this is not the case,

the two function calls, *sendto* and *recvfrom*, need to be repeated. However, the server is not aware of multidatagram communication; it handles each request separately.

### Example 17.2

As an example, let us see how we can design and write two programs: an *echo* server and an *echo* server. The client sends a line of text to the server; the server sends the same line back to the client. Although this client/server pair looks useless, it has some applications. It can be used, for example, when a computer wants to test if another computer in the network is alive. To better understand the code in a program, we first give the layout of variables used in both programs as shown in Figure 17.10.

**Figure 17.10** *Variables used in echo server and echo client using UDP service*

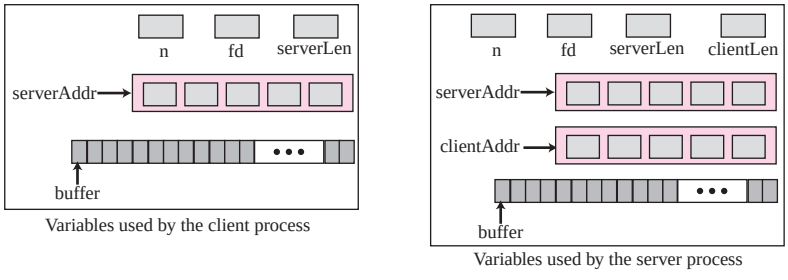



Table 17.1 shows the program for the echo server. We have eliminated many details and error-handling to simplify the program. We ask the reader to provide more details in exercises.

**Table 17.1** *Echo Server Program using the Service of UDP*

```
01 // UDP echo server program
02 #include "headerFiles.h"
03
04 int main (void)
05 {
06     // Declaration and definition
07     int sd;                              // Socket descriptor
08     int nr;                              // Number of bytes received
09     char buffer [256];                   // Data buffer
10     struct sockaddr_in serverAddr;       // Server address
11     struct sockaddr_in clientAddr;       // Client address
12     int clAddrLen;                       // Length of client Address
13     // Create socket
14     sd = socket (PF_INET, SOCK_DGRAM, 0);
15     // Bind socket to local address and port
16     memset (&serverAddr, 0, sizeof (serverAddr));
17     serverAddr.sin_family = AF_INET;
```

**Table 17.1** *Echo Server Program using the Service of UDP (continued)*

```
18      serverAddr.sin_addr.s_addr = htonl (INADDR_ANY);        // Default address
19      serverAddr.sin_port = htons (7)     // We assume port 7
20      bind (sd, (struct sockaddr*) &serverAddr, sizeof (serverAddr));
21      // Receiving and echoing datagrams
22      for ( ; ; )         // Run forever
23      {
24          nr = recvfrom (sd, buffer, 256, 0, (struct sockaddr*)&clientAddr, &clAddrLen);
25          sendto (sd, buffer, nr, 0, (struct sockaddr*)&clientAddr, sizeof(clientAddr));
26      }
27  } // End of echo server program
```

Line 14 creates a socket. Lines 16 to 19 create the local socket address; the remote socket address will be created in line 24 as explained later. Line 20 binds the socket to the local socket address. Lines 22 to 26 receive and send datagrams, possibly from many clients. When the server receives a datagram from a client, the client socket address and the length of the socket address are returned to the server. These two pieces of information are used in line 34 to send the echo message to the corresponding client. Note that we have eliminated many error-checking lines of codes; they have left as exercises.

Table 17.2 shows the client program for an echo process. We have assumed that the client sends only one datagram to be echoed by the server. If we need to send more than one datagram, the data transfer sections should be repeated using a loop.

**Table 17.2** *Echo Client Program using the Service of UDP*

```
01  // UDP echo client program
02  #include "headerFiles.h"
04
04  int main (void)
05  {
06      // Declaration and definition
07      int sd;                          // Socket descriptor
08      int ns;                          // Number of bytes send
09      int nr;                          // Number of bytes received
10      char buffer [256];               // Data buffer
11      struct sockaddr_in serverAddr;   // Socket address
11      // Create socket
12      sd = socket (PF_INET, SOCK_DGRAM, 0);
13      // Create server socket address
14      memset (&servAddr, 0, sizeof(serverAddr));
15      servAddr.sin_family = AF_INET;
16      inet_pton (AF_INET, "server address", &serverAddr.sin_addr);
17      serverAddr.sin_port = htons (7);
18      // Send and receive datagrams
19      fgets (buffer, 256, stdin);
```

**Table 17.2** *Echo Client Program using the Service of UDP (continued)*

```
20      ns = sendto (sd, buffer, strlen (buffer), 0,
21            (struct sockaddr)&serverAddr, sizeof(serveraddr));
22      recvfrom (sd, buffer, strlen (buffer), 0, NULL, NULL);
23      buffer [nr] = 0;
24      printf ("Received from server:");
25      fputs (buffer, stdout);
26      // Close and exit
27      close (sd);
28      exit (0);
29  } // End of echo client program
```

Line 12 creates a socket. Lines 14 to 17 show how we create the server socket address; there is no need to create the client socket address. Lines 19 to 25 read a string from the keyboard, send it to the server, and receive it back. Line 23 adds a null character to the received string to make it displayable in line 25. In line 27, we close the socket. We have eliminated all error checking; we leave them as exercises.

**To be complete, error-checking code needs to be added to both server and client programs.**

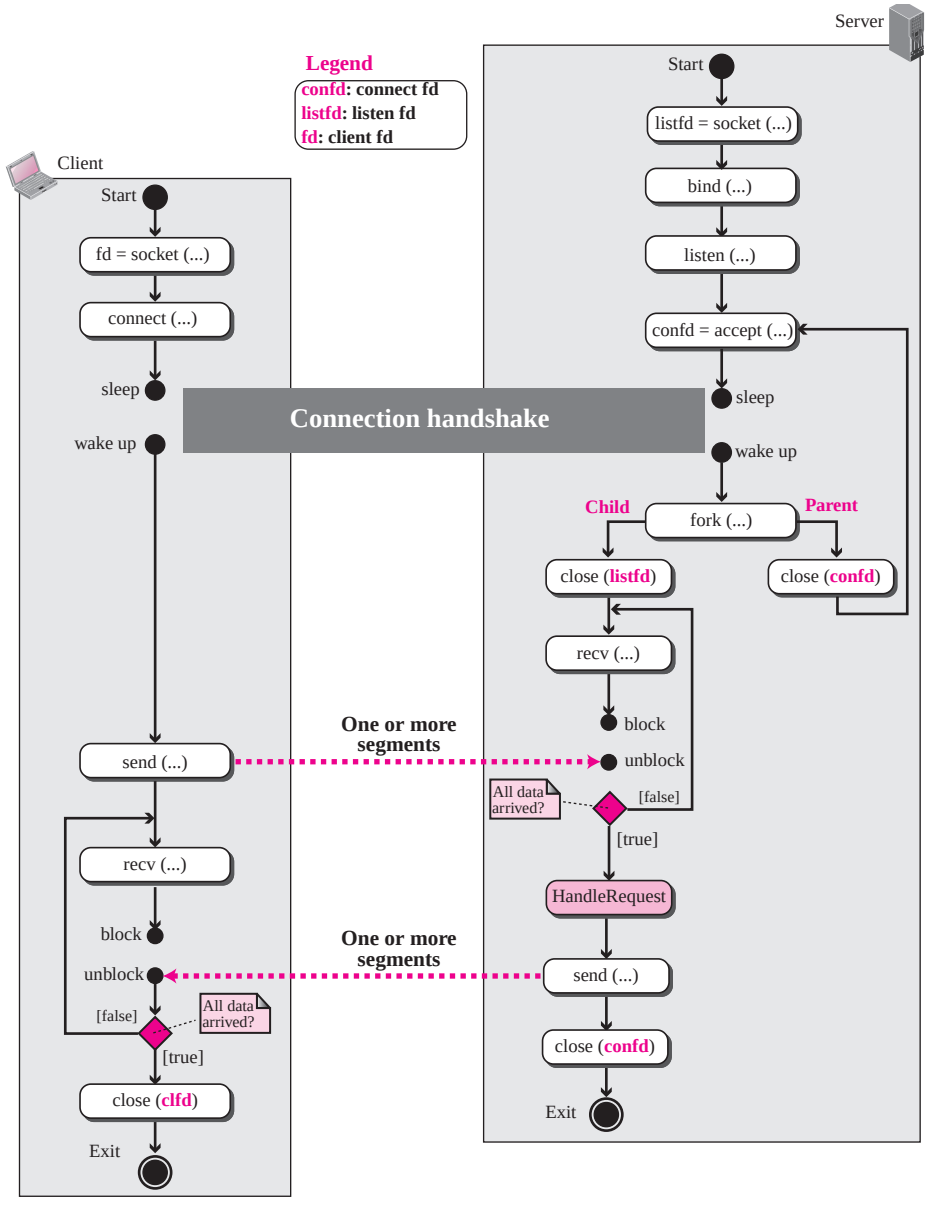
## Communication Using TCP

Now we discuss connection-oriented, concurrent communication using the service of TCP (the case of SCTP would be similar). Figure 17.11 shows the general flow diagram for this type of communication.

### *Server Process*

The server process starts first. It calls the *socket* function to create a socket, which we call the *listen* socket. This socket is only used during connection establishment. The server process then calls the *bind* function to bind this connection to the socket address of the server computer. The server program then calls the *accept* function. This function is a blocking function; when it is called, it is blocked until the TCP receives a connection request (SYN segment) from a client. The *accept* function then is unblocked and creates a new socket called the connect socket that includes the socket address of the client that sent the SYN segment. After the *accept* function is unblocked, the server knows that a client needs its service. To provide concurrency, the server process (parent process) calls the *fork* function. This function creates a new process (child process), which is exactly the same as the parent process. After calling the *fork* function, the two processes are running concurrently, but each can do different things. Each process now has two sockets: listen and connect sockets. The parent process entrusts the duty of serving the client to the hand of the child process and calls the accept function again to wait for another client to request connection. The child process is now ready to serve the client. It first closes the listen socket and calls the *recv* function to receive data from the client. The *recv* function, like the *recvfrom* function, is a blocking

**Figure 17.11** *Flow diagram for connection-oriented, concurrent communication*

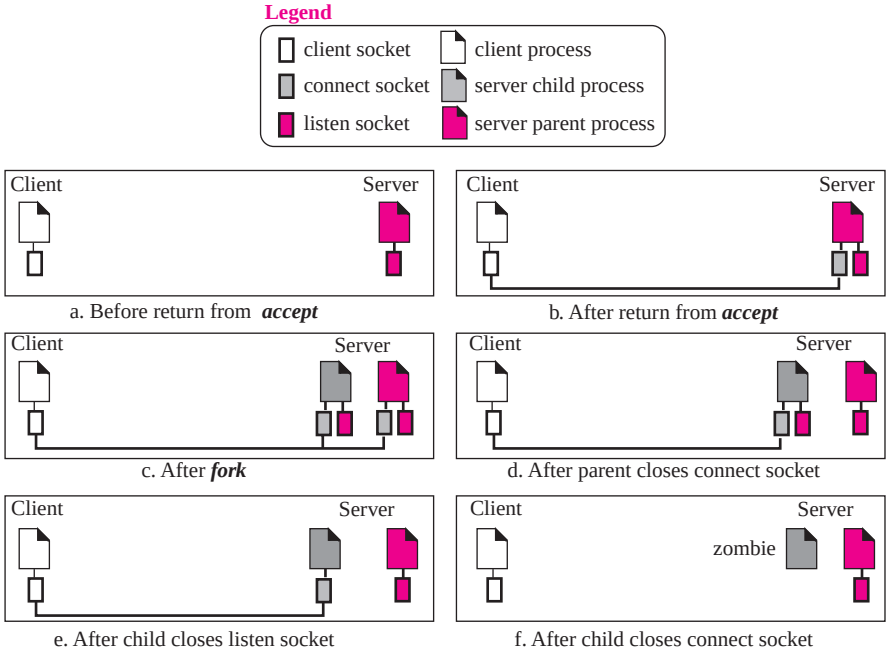



function; it is blocked until a segment arrives. The child process uses a loop and calls the *recv* function repeatedly until it receives all segments sent by the client. The child process then gives the whole data to a function (we call it *handleRequest*), to handle the request and return the result. The result is then sent to the client in one single call to the *send* function. We need to emphasize several points here. First, the flow diagram we are using

is the simplest possible one. The server may use many other functions to receive and send data, choosing the one which is appropriate for a particular application. Second, we assume that size of data to be sent to the client is so small that can be sent in one single call to the *send* function; otherwise, we need a loop to repeatedly call the *send* function. Third, although the server may send data using one single call to the *send* function, TCP may use several segments to send the data. The client process, therefore, may not receive data in one single segment, as we will see when we explain the client process.

Figure 17.12 shows the status of the parent and child process with respect to the sockets. Part a in the figure shows the status before the *accept* function returns. The parent process uses the *listen* socket to wait for request from the clients. When the accept function is blocked and returned (part b), the parent process has two sockets: the *listen* and the *connect* sockets. The client is connected to the *connect* socket. After calling the *fork* function (part c), we have two processes, each with two sockets. The client is connected to both processes.The parent needs to close its *connect* socket to free itself from the client and be free to listen to requests from other clients (part d). Before the child can start serving the connected client, it needs to close its *listen* socket so that a future request does not affect it (part e). Finally, when the child finishes serving the connected client, it needs to close its connect socket to disassociate itself from the client that has been served (part f).

**Figure 17.12** *Status of parent and child processes with respect to the sockets*



Although we do not include the operation in our program (for simplicity), the child process, after serving the corresponding process, needs to be destroyed. The child process that has done its duty and is dormant is normally called a zombie in the UNIX environment system. A child can be destroyed as soon as it is not needed. Alternatively, the system can run a special program once in a while to destroy all zombies in the system. The zombies occupy space in the system and can affect the performance of the system.

### *Client Process*

The client process is simpler. The client calls the *socket* function to create a socket. It then calls the *connect* function to request a connection to the server. The *connect* function is a blocking function; it is blocked until the connection is established between two TCPs. When the *connect* function returns, the client calls the *send* function to send data to the server. We use only one call to the *send* function, assuming that data can be sent with one call. Based on the type of the application, we may need to call this function repeatedly (in a loop). The client then calls the *recv* function, which is blocked until a segment arrives and data are delivered to the process by TCP. Note that, although the data are sent by the server in one single call to the *send* function, the TCP at the server site may have used several segments to send data. This means we may need to call the *recv* function repeatedly to receive all data. The loop can be controlled by the return value of the *recv* function.

### Example 17.3

We want to write two programs to show how we can have an echo client and echo server using the services of TCP. Figure 17.13 shows the variables we use in these two programs. Since data may arrive in different chunks, we need pointers to point to the buffer. The first buffer is fixed and always points to the beginning of the buffer; the second pointer is moving to let the arrived bytes be appended to the end of the previous section.

**Figure 17.13** *Variable used Example 17.3.*

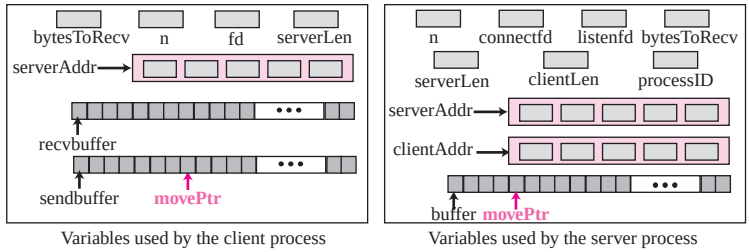



Table 17.3 shows the server program for an echo server that uses the services of TCP. We have eliminated many details and left them for the books on the network programming. We want just to show a global picture of the program.

**Table 17.3** *Echo Server Program using the Services of TCP*

```
01 // Echo server program
02 #include "headerFiles.h"
03
04 int main (void)
05 {
06      // Declaration and definition
07      int listensd;                          // Listen socket descriptor
08      int connectsd;                         // Connecting socket descriptor
09      int n;                                 // Number of bytes in each reception
10      int  bytesToRecv;                      // Total bytes to receive
11      int   processID;                       // ID of the child process
12      char buffer [256];                     // Data buffer
13      char* movePtr;                         // Pointer to the buffer
14      struct sockaddr_in serverAddr;         // Server address
15      struct sockaddr_in clientAddr;         // Client address
16       int clAddrLen;                        // Length of client address
17      // Create listen socket
18      listensd = socket (PF_INET, SOCK_STREAM, 0);
19      // Bind listen socket to the local address and port
20      memset (&serverAddr, 0, sizeof (serverAddr));
21      serverAddr.sin_family = AF_INET;
22      serverAddr.sin_addr.s_addr = htonl (INADDR_ANY);
23      serverAddr.sin_port = htons (7);  // We assume port 7
24      bind (listensd, &serverAddr, sizeof (serverAddr));
25      // Listen to connection requests
26       listen (listensd, 5);
27      // Handle the connection
28       for ( ; ; )                    // Run forever
29      {
30           connectsd = accept (listensd, &clientAddr, &clAddrLen);
31           processID = fork ();
32           if (processID == 0)                    // Child process
33           {
34                close (listensd);
35                bytesToRecv = 256;
36                movePtr = buffer;
37                while ( (n = recv (connectfd, movePtr, bytesToRecv, 0)) > 0)
38                {
39                     movePtr = movePtr + n;
40                     bytesToRecv = movePtr – n;
41                }  // End of while
```

**Table 17.3** *Echo Server Program using the Services of TCP (continued)*

```
42              send (connectsd, buffer, 256, 0);
43              exit (0);
44          } // End of if
45          close (connectsd):                    // Back to parent process
46      } // End of for loop
47  } // End of echo server program
```

The program follows the flow diagram of Figure 17.11. Every time the *recv* function is unblocked it gets some data and stores it at the end of the buffer. The movePtr is then moved to point to the location where the next data chunk should be stored (line 39). The number of bytes to read is also decremented from the original value (26) to prevent overflow in the buffer (line 40). After all data have been received, the server calls the *send* function to send them to the client. As we mentioned before, there is only one *send* call, but TCP may send data in several segments. The child process then calls the *close* function to destroy the connect socket.

Table 17.4 shows the client program for the echo process that uses the services of TCP. It then uses the same strategy described in Table 17.2 to create the server socket address. The program then gets the string to be echoed from the keyboard, stores it in *sendBuffer*, and sends it. The result may come in different segments. The program uses a loop and repeat, calling the *recv* function until all data arrive. As usual, we have ignored code for error checking to make the program simpler. It needs to be added if the code is actually used to send data to a server.

**Table 17.4** *Echo Client Program using the services of TCP*

```
01  // TCP echo client program
02  #include "headerFiles.h"
03
04  int main (void)
05  {
06      // Declaration and definition
07      int  sd;                              // Socket descriptor
08      int  n;                               // Number of bytes received
09      int  bytesToRecv;                     // Number of bytes to receive
10      char  sendBuffer [256];               // Send buffer
11      char  recvBuffer [256];               // Receive buffer
12      char* movePtr;                        // A pointer the received buffer
13      struct sockaddr_in  serverAddr;       // Server address
14
15      // Create socket
16      sd = socket (PF_INET, SOCK_STREAM, 0);
17      // Create server socket address
18      memset (&serverAddr, 0, sizeof(serverAddr);
19      serverAddr.sin_family = AF_INET;
20      inet_pton (AF_INET, "server address", &serverAddr.sin_addr);
21      serverAddr.sin_port = htons (7);    // We assume port 7
22      // Connect
```

**Table 17.4** *Echo Client Program using the services of TCP (continued)*

```
23      connect (sd, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
24      // Send and receive data
25      fgets (sendBuffer, 256, stdin);
26      send (fd, sendBuffer, strlen (sendbuffer), 0);
27      bytesToRecv = strlen (sendbuffer);
28      movePtr = recvBuffer;
29      while ( (n = recv (sd, movePtr, bytesToRecv, 0) ) > 0)
30      {
31          movePtr = movePtr + n;
32          bytesToRecv = bytesToRecv – n;
33      } // End of while loop
34      recvBuffer[bytesToRecv] = 0;
35      printf ("Received from server:");
36      fputs (recvBuffer, stdout);
37      // Close and exit
38      close (sd);
39      exit (0);
40  } // End of echo client program
```

### Predefined Client-Server Applications

The Internet has defined a set of applications using **client-server paradigms.** They are established programs that can be installed and be used. Some of these applications are designed to give some specific service (such as FTP), some are designed to allow users to log into the server and perform the desired task (such TELNET), and some are designed to help other application programs (such as DNS). We discuss these application programs in detail in Chapters 18 to 24.

**In Appendix F we give some simple Java versions of programs in Table 17.1 to 17.4**

## 17.2 PEER-TO-PEER PARADIGM

Although most of the applications available in the Internet today use the client-server paradigm, the idea of using the so called **peer-to-peer (P2P) paradigm** recently has attracted some attention. In this paradigm, two peer computers (laptops, desktops, or main frames) can communicate with each other to exchange services. This paradigm is interesting in some areas such file as transfer in which the client-server paradigm may put a lot of the load on the server machine if a client wants to transfer a large file such as an audio or video file. The idea is also interesting if two peers need to exchange some files or information to each other without going to a server. However, we need to mention that the P2P paradigm does not ignore the client-server paradigm. What it does

actually is to let the duty of a server be shared by some users that want to participate in the process. For example, instead of allowing several clients to make a connection and each download a large file, a server can let each client download a part of a file and then share it with each other. In the process of downloading part of the file or sharing the downloaded file, however, a computer needs to play the role of a client and the other the role of a server. In other words, a computer can be a client for a specific application at one moment and the server at another moment. These applications are now controlled commercially and not formally part of the Internet. We leave the exploration of these applications to the books designed for each specific application.

## 17.3 FURTHER READING

Several books give thorough coverage of network programming. In particular, we recommend [Ste et al. 04], [Com 96], [Rob & Rob 96], and [Don & Cal 01].

## 17.4 KEY TERMS

client-server paradigm
interface
peer-to-peer (P2P) paradigm
socket
socket interface
STREAM
transport-layer interface (TLI)

## 17.5 SUMMARY

- Most applications in the Internet are designed using a client-server paradigm in which an application program, called a server, provides services and another application program, called a client, receives services. A server program is an infinite program. When it starts it runs infinitely unless a problem arises. It waits for incoming requests from clients. A client program is finite, which means it is started by the user and terminates when the service is complete. Both clients and servers can run in concurrent mode.
- Clients can be run on a machine either iteratively or concurrently. An iterative server can process only one request at a time. A concurrent server, on the other hand, can process many requests at the same time.
- Client-server paradigm is based on a set of predefined functions called an interface. We discussed the most common interface, called socket interface, in this chapter. The socket interface, as a set of instructions, is located between the operating stem and the application programs. A socket is a software abstract simulating the hardware socket we see in our daily life. To use the communication channel, an application program (client or server) needs to request the operating system to create a socket.

- The interaction between a process and the operating system is done through a list of predefined functions. In this chapter, we introduced a subset of these functions, namely *socket*, *bind*, *connect*, *listen*, *accept*, *fork*, *send*, *recv*, *sendto*, *recvfrom*, and *close*. We also introduced some byte ordering functions and some memory management functions.
- A server can be designed to be a connectionless iterative program using the services of UDP or a connection-oriented concurrent program using the services of TCP or SCTP.
- Although the client-server paradigm is very common in the Internet today, another paradigm, called peer-to-peer paradigm, has found some commercial use.

## 17.6 PRACTICE SET

### Exercises

1. Show how a 32-bit integer is stored in four memory locations (bytes $x$, $x + 1$, $x + 2$, and $x + 3$) using the little-endian byte order.
2. Show how a 32-bit integer is stored in four memory locations (bytes $x$, $x + 1$, $x + 2$, and $x + 3$) using the little-endian byte order.
3. Write a short program to test if your computer is using big-endian or little-endian byte order.
4. We have used several data types in the programs in this chapter. Write a short program to use and test them.
5. Write a short program to test memory functions we used in this chapter.
6. There are several functions in UNIX that can be used to retrieve host information, such as the IP address and port number—for example, *gethostbyname*, *gethostbyaddress*, *getservebyname*, and *getservebyport*. Try to find some information about these functions and use them in a short program.
7. In Table 17.1, add some code to check if there is an error in the call to the *socket* function. The program needs to exit if an error occurs. Hint: use the *perror* and *exit* functions. The prototype for the *perror* function is shown below:

```
void perror (const char* string);
```

8. In Table 17.1, add some code to check if there is an error in the call to bind function. The program needs to exit if an error occurs. Hint: use the *perror* and *exit* functions.
9. Modify the program in Table 17.2 to allow the client to send more than one datagram.
10. Modify the flow diagram in Figure 17.11 to show a connectionless but iterative server.

CHAPTER

# 18 Host Configuration: DHCP

In this chapter we discuss our first client/server application program, Dynamic Host Configuration Protocol (DHCP). This application is discussed first because it is the first client/server application program that is used after a host is booted. In other words, it serves as a bootstrap when a host is booted and supposed to be connected to the Internet, but the host does not know its IP address.

## OBJECTIVES

*The chapter has several objectives:*

- To give the reasons why we need host configuration.
- To give a historical background of two protocols used for host configuration in the past.
- To define DHCP as the current Dynamic Host Configuration Protocol.
- To discuss DHCP operation when the client and server are on the same network or on different networks.
- To show how DHCP uses two well-known ports of UDP to achieve configuration.
- To discuss the states the clients go through to lease an IP address from a DHCP server.

## 18.1 INTRODUCTION

Each computer that uses the TCP/IP protocol suite needs to know its IP address. If the computer uses classless addressing or is a member of a subnet, it also needs to know its subnet mask. Most computers today need two other pieces of information: the address of a default router to be able to communicate with other networks and the address of a name server to be able to use names instead of addresses as we will see in the next chapter. In other words, four pieces of information are normally needed:

1. The IP address of the computer
2. The subnet mask of the computer
3. The IP address of a router
4. The IP address of a name server

These four pieces of information can be stored in a configuration file and accessed by the computer during the bootstrap process. But what about a diskless workstation or a computer with a disk that is booted for the first time?

In the case of a diskless computer, the operating system and the networking software could be stored in read-only memory (ROM). However, the above information is not known to the manufacturer and thus cannot be stored in ROM. The information is dependent on the individual configuration of the machine and defines the network to which the machine is connected.

### Previous Protocols

Before DHCP became the formal protocol for host configuration, some other protocols were used for this propose. We briefly describe them here.

#### *RARP*

At the beginning of the Internet era, a protocol called Reverse Address Resolution Protocol (RARP) was designed to provide the IP address for a booted computer. RARP was actually a version of ARP that we discussed in Chapter 8. ARP maps an IP address to a physical address: RARP maps a physical address to an IP address. However, RARP is deprecated today for two reasons. First, RARP used the broadcast service of the data link layer, which means that a RARP server must be present in each network. Second, RARP can provide only the IP address of the computer, but a computer today needs all four pieces of information mentioned above.

#### *BOOTP*

The **Bootstrap Protocol (BOOTP)** is the prerunner of DHCP. It is a client/server protocol designed to overcome the two deficiencies of the RARP protocol. First, since it is a client/server program, the BOOTP server can be anywhere in the Internet. Second, it

can provide all pieces of information we mentioned above, including the IP address. To provide the four pieces of information described above, it removes all restriction about the RARP protocol. BOOTP, however, is a *static configuration protocol*. When a client requests its IP address, the BOOTP server consults a table that matches the physical address of the client with its IP address. This implies that the binding between the physical address and the IP address of the client already exists. The binding is predetermined.

There are some situations in which we need a *dynamic configuration protocol*. For example, when a host moves from one physical network to another, its physical address changes. As another example, there are occasions when a host wants a temporary IP address to be used for a period of time. BOOTP cannot handle these situations because the binding between the physical and IP addresses is static and fixed in a table until changed by the administrator. As we will see shortly, DHCP has been devised to handle these shortcomings.

### DHCP

The **Dynamic Host Configuration Protocol (DHCP)** is a client/server protocol designed to provide the four pieces of information for a diskless computer or a computer that is booted for the first time. DHCP is a successor to BOOTP and is backward compatible with it. Although BOOTP is considered deprecated, there may be some systems that may still use BOOTP for host configuration. The part of the discussion in this chapter that does not deal with the dynamic aspect of DHCP can also be applied to BOOTP.

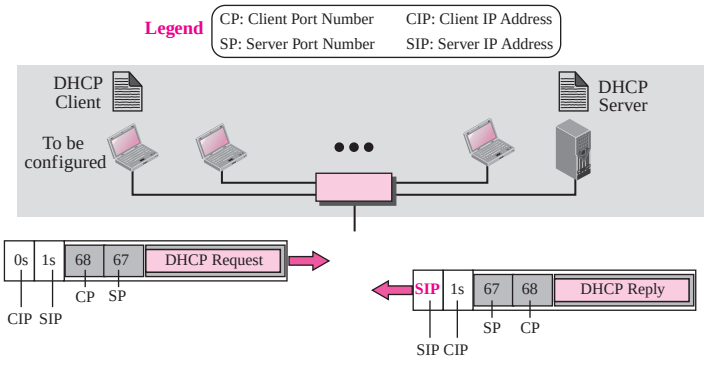
## 18.2 DHCP OPERATION

The DHCP client and server can either be on the same network or on different networks. Let us discuss each situation separately.

### Same Network

Although the practice is not very common, the administrator may put the client and the server on the same network as shown in Figure 18.1.

**Figure 18.1** *Client and server on the same network*



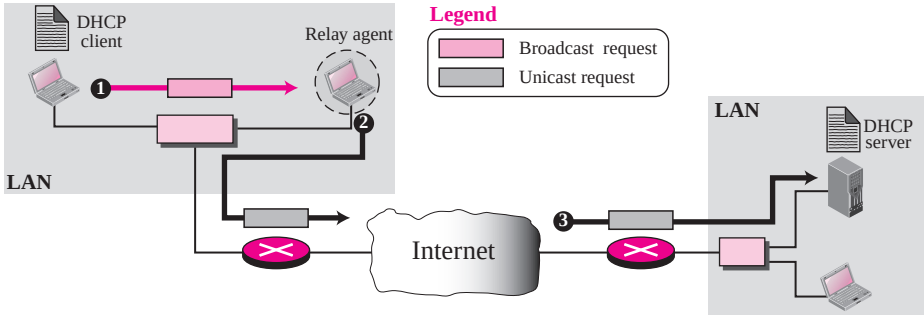
In this case, the operation can be described as follows:

1. The DHCP server issues a passive open command on UDP port number 67 and waits for a client.
2. A booted client issues an active open command on port number 68 (this number will be explained later). The message is encapsulated in a UDP user datagram, using the destination port number 67 and the source port number 68. The UDP user datagram, in turn, is encapsulated in an IP datagram. The reader may ask how a client can send an IP datagram when it knows neither its own IP address (the source address) nor the server's IP address (the destination address). The client uses all 0s as the source address and all 1s as the destination address.
3. The server responds with either a broadcast or a unicast message using UDP source port number 67 and destination port number 68. The response can be unicast because the server knows the IP address of the client. It also knows the physical address of the client, which means it does not need the services of ARP for logical to physical address mapping. However, some systems do not allow the bypassing of ARP, resulting in the use of the broadcast address.

## Different Networks

As in other application-layer processes, a client can be in one network and the server in another, separated by several other networks. Figure 18.2 shows the situation.

**Figure 18.2** *Client and server on two different networks*




However, there is one problem that must be solved. The DHCP request is broadcast because the client does not know the IP address of the server. A broadcast IP datagram cannot pass through any router. A router receiving such a packet discards it. Recall that an IP address of all 1s is a limited broadcast address.

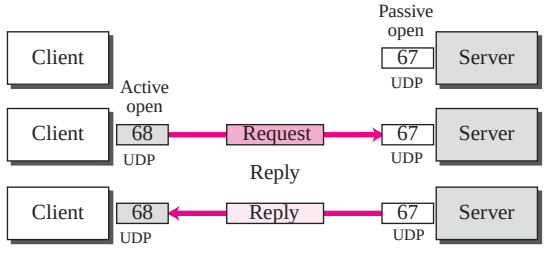
To solve the problem, there is a need for an intermediary. One of the hosts (or a router that can be configured to operate at the application layer) can be used as a relay. The host in this case is called a **relay agent.** The relay agent knows the unicast address of a DHCP server and listens for broadcast messages on port 67. When it receives this type of packet, it encapsulates the message in a unicast datagram and sends the request to the DHCP server. The packet, carrying a unicast destination address, is routed by any

router and reaches the DHCP server. The DHCP server knows the message comes from a relay agent because one of the fields in the request message defines the IP address of the relay agent. The relay agent, after receiving the reply, sends it to the DHCP client.

## UDP Ports

Figure 18.3 shows the interaction between a client and a DHCP server. The server uses the well-known port 67, which is normal. The client uses the well-known port 68, which is unusual. The reason for choosing the well-known port 68 instead of an ephemeral port is to prevent a problem when the reply, from the server to the client, is broadcast. To understand the problem, let us look at a situation where an ephemeral port is used. Suppose host A on a network is using a DHCP client on ephemeral port 2017 (randomly chosen). Host B, on the same network, is using a DAYTIME client on ephemeral port 2017 (accidentally the same). Now the DHCP server sends a broadcast reply message with the destination port number 2017 and broadcast IP address $FFFFFFFF_{16}$. Every host needs to open a packet carrying this destination IP address. Host A finds a message from an application program on ephemeral port 2017. A correct message is delivered to the DHCP client. An incorrect message is delivered to the DAYTIME client. The confusion is due to the demultiplexing of packets based on the socket address (see Chapter 17), which is a combination of IP address and port number. In this case, both are the same.

**Figure 18.3** *Use of UDP ports*




The use of a well-known port (less than 1024) prevents the use of the same two destination port numbers. Host B cannot select 68 as the ephemeral port because ephemeral port numbers are greater than 1023.

The curious reader may ask what happens if host B is also running the DHCP client. In this case, the socket address is the same and both clients will receive the message. In this situation, a third identification number differentiates the clients. DHCP uses another number, called the transaction ID, which is randomly chosen for each connection involving DHCP. It is highly improbable that two hosts will choose the same ID at the same time.

## Using TFTP

The server does not send all of the information that a client may need for booting. In the reply message, the server defines the pathname of a file in which the client can find

complete booting information. The client can then use a TFTP message (see Chapter 21), which is encapsulated in a UDP user datagram, to obtain the rest of the needed information.

## Error Control

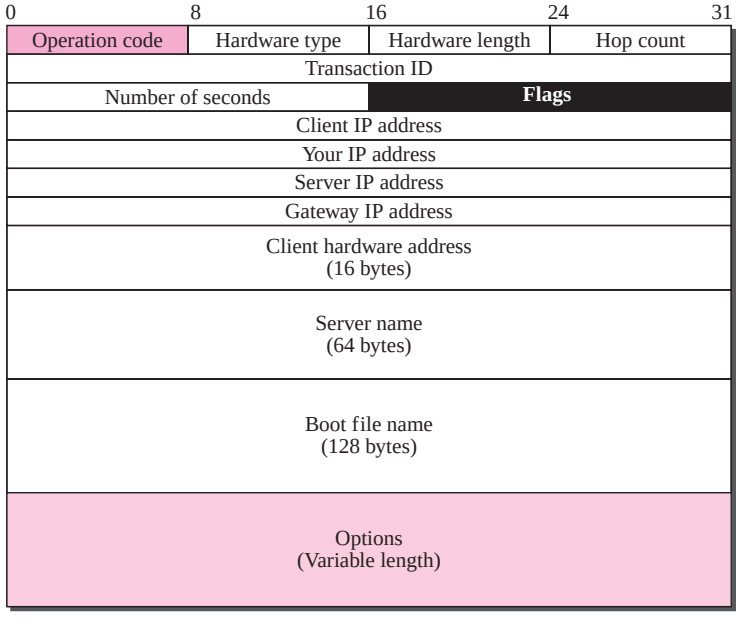
What if a request is lost or damaged? What if the response is damaged? There is a need for error control when using DHCP. DHCP uses UDP, which does not provide error control. Therefore, DHCP must provide error control. Error control is accomplished through two strategies:

1. DHCP requires that UDP uses the checksum. Remember that the use of the checksum in UDP is optional.
2. The DHCP client uses timers and a retransmission policy if it does not receive the DHCP reply to a request. However, to prevent a traffic jam when several hosts need to retransmit a request (for example, after a power failure), DHCP forces the client to use a random number to set its timers.

## Packet Format

Figure 18.4 shows the format of a DHCP packet.

**Figure 18.4** *DHCP packet format*

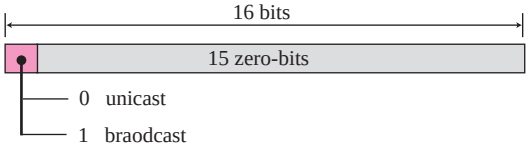



The following briefly describes each field:

- **Operation code.** This 8-bit field defines the type of DHCP packet: request (1) or reply (2).

- **Hardware type.** This is an 8-bit field defining the type of physical network. Each type of network has been assigned an integer. For example, for Ethernet the value is 1.
- **Hardware length.** This is an 8-bit field defining the length of the physical address in bytes. For example, for Ethernet the value is 6.
- **Hop count.** This is an 8-bit field defining the maximum number of hops the packet can travel.
- **Transaction ID.** This is a 4-byte field carrying an integer. The transaction identification is set by the client and is used to match a reply with the request. The server returns the same value in its reply.
- **Number of seconds.** This is a 16-bit field that indicates the number of seconds elapsed since the time the client started to boot.
- **Flag.** This is a 16-bit field in which only the leftmost bit is used and the rest of the bits should be set to 0s. A leftmost bit specifies a forced broadcast reply (instead of unicast) from the server. If the reply were to be unicast to the client, the destination IP address of the IP packet is the address assigned to the client. Since the client does not know its IP address, it may discard the packet. However, if the IP datagram is broadcast, every host will receive and process the broadcast message. Figure 18.5 shows the flag format.

**Figure 18.5** *Flag format*

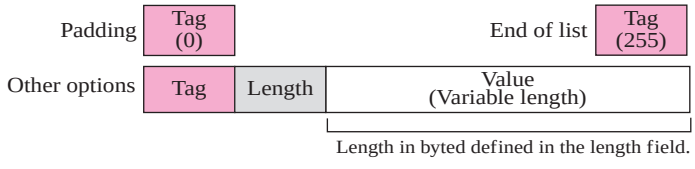



- **Client IP address.** This is a 4-byte field that contains the client IP address. If the client does not have this information, this field has a value of 0.
- **Your IP address.** This is a 4-byte field that contains the client IP address. It is filled by the server (in the reply message) at the request of the client.
- **Server IP address.** This is a 4-byte field containing the server IP address. It is filled by the server in a reply message.
- **Gateway IP address.** This is a 4-byte field containing the IP address of a router. It is filled by the server in a reply message.
- **Client hardware address.** This is the physical address of the client. Although the server can retrieve this address from the frame sent by the client, it is more efficient if the address is supplied explicitly by the client in the request message.
- **Server name.** This is a 64-byte field that is optionally filled by the server in a reply packet. It contains a null-terminated string consisting of the domain name of the

server. If the server does not want to fill this field with data, the server must fill it with all 0s.

- **Boot filename.** This is a 128-byte field that can be optionally filled by the server in a reply packet. It contains a null-terminated string consisting of the full pathname of the boot file. The client can use this path to retrieve other booting information. If the server does not want to fill this field with data, the server must fill it with all 0s.
- **Options.** This is a 64-byte field with a dual purpose. It can carry either additional information (such as the network mask or default router address) or some specific vendor information. The field is used only in a reply message. The server uses a number, called a **magic cookie,** in the format of an IP address with the value of 99.130.83.99. When the client finishes reading the message, it looks for this magic cookie. If present, the next 60 bytes are options. An option is composed of three fields: a 1-byte tag field, a 1-byte length field, and a variable-length value field. The length field defines the length of the value field, not the whole option. See Figure 18.6.

**Figure 18.6** *Option format*




The list of options is shown in Table 18.1.

**Table 18.1** *Options for DHCP*

| *Tag* | *Length* | *Value* | *Description* |
|---|---|---|---|
| 0 | | | Padding |
| 1 | 4 | Subnet mask | Subnet mask |
| 2 | 4 | Time of the day | Time offset |
| 3 | Variable | IP addresses | Default router |
| 4 | Variable | IP addresses | Time server |
| 5 | Variable | IP addresses | IEN 16 server |
| 6 | Variable | IP addresses | DNS server |
| 7 | Variable | IP addresses | Log server |
| 8 | Variable | IP addresses | Quote server |
| 9 | Variable | IP addresses | Print server |
| 10 | Variable | IP addresses | Impress |
| 11 | Variable | IP addresses | RLP server |
| 12 | Variable | DNS name | Host name |
| 13 | 2 | Integer | Boot file size |
| 53 | 1 | Discussed later | Used for dynamic configuration |
| 128–254 | Variable | Specific information | Vendor specific |
| 255 | | | End of list |

The lengths of the fields that contain IP addresses are multiples of 4 bytes. The padding option, which is only 1 byte long, is used only for alignment. The end-of-list option, which is also only 1 byte long, indicates the end of the option field. Vendors can use option tags 128 to 254 to supply extra information in a reply message.

## 18.3 CONFIGURATION

The DHCP has been devised to provide static and dynamic address allocation.

### Static Address Allocation

In this capacity, a DHCP server has a database that statically binds physical addresses to IP addresses. When working in this way, DHCP is backward compatible with the deprecated protocol BOOTP, which we discussed before.

### Dynamic Address Allocation

DHCP has a second database with a pool of available IP addresses. This second database makes DHCP dynamic. When a DHCP client requests a temporary IP address, the DHCP server goes to the pool of available (unused) IP addresses and assigns an IP address for a negotiable period of time.

When a DHCP client sends a request to a DHCP server, the server first checks its static database. If an entry with the requested physical address exists in the static database, the permanent IP address of the client is returned. On the other hand, if the entry does not exist in the static database, the server selects an IP address from the available pool, assigns the address to the client, and adds the entry to the dynamic database.

The dynamic aspect of DHCP is needed when a host moves from network to network or is connected and disconnected from a network (for example, a subscriber to a service provider). DHCP provides temporary IP addresses for a limited period of time.

The addresses assigned from the pool are temporary addresses. The DHCP server issues a **lease** for a specific period of time. When the lease expires, the client must either stop using the IP address or renew the lease. The server has the choice to agree or disagree with the renewal. If the server disagrees, the client stops using the address.

### Transition States

To provide dynamic address allocation, the DHCP client acts as a state machine that performs transitions from one state to another depending on the messages it receives or sends. The type of the message in this case is defined by the option with tag 53 that is included in the DHCP packet. In other words, instead of adding one extra field to the BOOTP protocol to define DHCP type, the designer decided to add an extra option for this purpose. Figure 18.7 shows the type option and the interpretation of its value to define the type of the DHCP packet.

Figure 18.8 shows the transition diagram with main states. The RFC and some implementations offer some more states that we leave the investigation as exercises.

**Figure 18.7** *Option with tag 53*

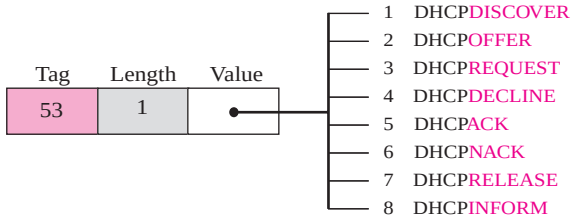
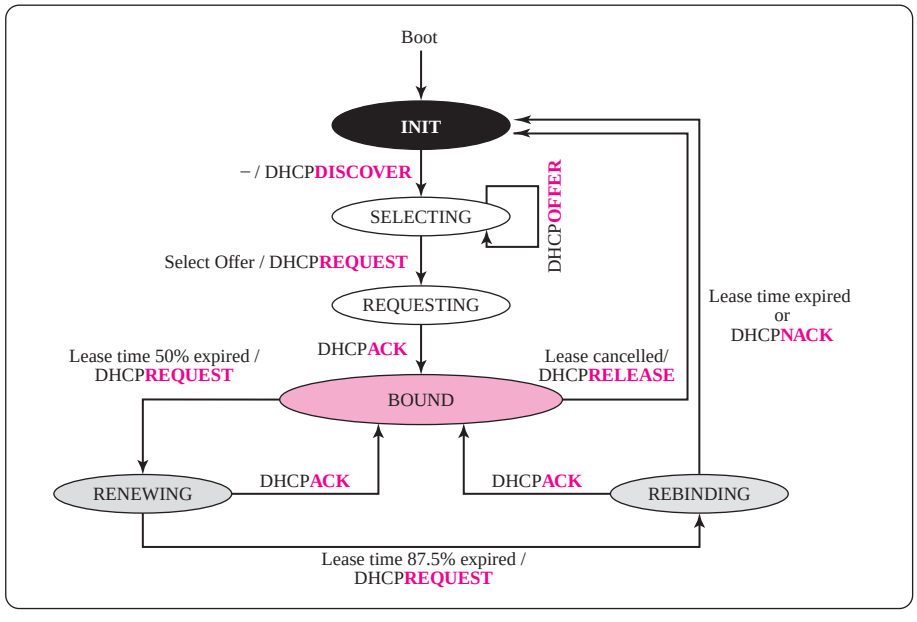



**Figure 18.8** *DHCP client transition diagram*




### *INIT State*

When the DHCP client first starts, it is in the INIT state (initializing state). The client broadcasts a DHCPDISCOVER message (a request message with the DHCPDISCOVER option), using port 67.

### *SELECTING State*

After sending the DHCPDISCOVER message, the client goes to the selecting state. Those servers that can provide this type of service respond with a DHCPOFFER message. In these messages, the servers offer an IP address. They can also offer the lease duration. The default is 1 hour. The server that sends a DHCPOFFER locks the offered IP address so that it is not available to any other clients. The client chooses one of the

offers and sends a DHCPREQUEST message to the selected server. It then goes to the requesting state. However, if the client receives no DHCPOFFER message, it tries four more times, each with a span of 2 seconds. If there is no reply to any of these DHCPDISCOVERs, the client sleeps for 5 minutes before trying again.

#### *REQUESTING State*

The client remains in the requesting state until it receives a DHCPACK message from the server that creates the binding between the client physical address and its IP address. After receipt of the DHCPACK, the client goes to the bound state.

#### *BOUND State*

In this state, the client can use the IP address until the lease expires. When 50 percent of the lease period is reached, the client sends another DHCPREQUEST to ask for renewal. It then goes to the renewing state. When in the bound state, the client can also cancel the lease and go to the initializing state.

#### *RENEWING State*

The client remains in the renewing state until one of two events happens. It can receive a DHCPACK, which renews the lease agreement. In this case, the client resets its timer and goes back to the bound state. Or, if a DHCPACK is not received, and 87.5 percent of the lease time expires, the client goes to the rebinding state.

#### *REBINDING State*

The client remains in the rebinding state until one of three events happens. If the client receives a DHCPNACK or the lease expires, it goes back to the initializing state and tries to get another IP address. If the client receives a DHCPACK, it goes to the bound state and resets the timer.

### Other Issues

In this section we discuss a few issues related to the DHCP states.

#### *Early Release*

A DHCP client that has been assigned an address for a period of time may release the address before the expiration time. The client may send a DHCPRELEASE message to tell the server that the address is no longer needed. This helps the server to assign the address to another client waiting for it.

#### *Timers*

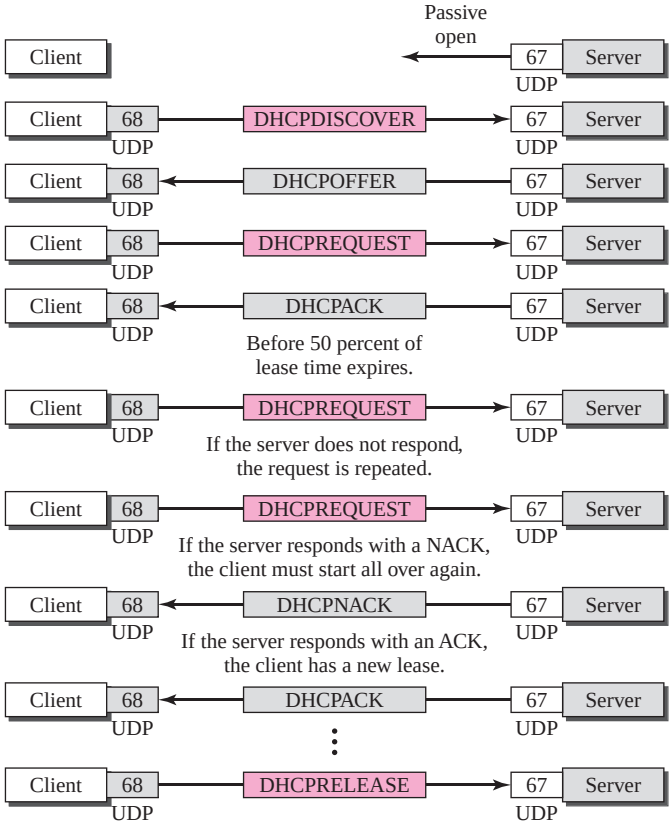
The above discussion requires that the client uses three times: renewal timer, rebinding timer, and expiration timer. If the server does not specify the time-out values for these timers when the address is allocated, the client needs to use the default value. The default value for each timer is shown below:

| | | |
|---|---|---|
| **Renewal timer:** | → | **50% of lease time** |
| **Rebinding timer:** | → | **87.5% of lease time** |
| **Expiration timer:** | → | **100% of lease time** |

### Exchanging Messages

Figure 18.9 shows the exchange of messages related to the transition diagram.

**Figure 18.9** *Exchanging messages*




## 18.4 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books and RFCs

Several books and RFCs give an easy but thorough coverage of DHCP including [Com 06], RFC 3396, and RFC 3342.

## 18.5 KEY TERMS

Bootstrap Protocol (BOOTP)
Dynamic Host Configuration Protocol (DHCP)
lease
magic cookie
relay agent

## 18.6 SUMMARY

- Every computer attached to a TCP/IP internet must know its IP address, the IP address of a router, the IP address of a name server, and its subnet mask. Dynamic Host Configuration Protocol (DHCP) is a client-server application that delivers vital network information to either diskless computers or computers at first boot.
- One DHCP packet format is used for both the client request and the server reply. The DHCP server waits passively for a client request. A server reply can be broadcast or unicast. A DHCP request or reply is encapsulated in a UDP user datagram.
- When the DHCP client and server are on different networks, a relay agent is used to send local DHCP requests from a client to remote servers.
- When DHCP acts as a static configuration protocol, it uses a table that maps IP addresses to physical addresses. When DHCP acts as a dynamic configuration protocol, it leases IP addresses to the requesting clients.
- DHCP client is designed as a state machine that uses six main states and three timers to allow a host to lease an IP address for a specified period of time.

## 18.7 PRACTICE SET

### Exercises

1. What is the minimum length of a DHCP packet? What is the maximum length?
2. A DHCP packet is encapsulated in a UDP packet, which is encapsulated in an IP packet, which is encapsulated in an Ethernet frame. Find the efficiency of a DHCP packet when no option is used. The efficiency in this case is measured in the number of bytes in the DHCP packet to the total number of bytes transmitted at the data link layer.
3. Show an example of a DHCP packet with a padding option.
4. Show an example of a DHCP packet with an end-of-list option.
5. What is the maximum number of seconds that can be stored in the Number of Seconds field of a DHCP packet?

6. **a.** Show the contents of all fields for a DHCP request packet sent from a client with physical address 00:11:21:15:EA:21.
   **b.** Encapsulate the packet in part a in a UDP user datagram. Fill all the fields.
   **c.** Encapsulate the packet in part b in an IP datagram. Fill all the fields.
   **d.** Show the contents of all fields for a DHCP reply sent in response to the request in part a.
   **e.** Encapsulate the packet in part d in a UDP user datagram. Fill all the fields.
   **f.** Encapsulate the packet in part e in an IP datagram. Fill all the fields.
7. Why does a newly added host need to know its subnet mask?
8. Why does a newly added host need to know the IP address of a router?
9. Why does a newly added host need to know the IP address of a name server?
10. Why do you think DHCP needs to use TFTP to get additional information? Why can't all the information be retrieved using DHCP?
11. A diskless client on a Class C Ethernet network uses DHCP. The DHCP server is on a Class B Ethernet network. Draw a figure of the networks with appropriate IP addresses for the client, server, and relay agent. Fill out a DHCP request and reply packet.

## Research Activities

12. Show the format and contents of a DHCPDISCOVER message.
13. Show the format and contents of a DHCPOFFER message.
14. Show the format and contents of a DHCPREQUEST message.
15. Show the format and contents of a DHCPDECLINE message.
16. Show the format and contents of a DHCPACK message.
17. Show the format and contents of a DHCPNACK message.
18. Show the format and contents of a DHCPRELEASE message.
19. Find the range of random numbers used to set timers that control the retransmission of lost DHCP packets.
20. Find if there is a limit for the number of times a client can retransmit a request.
21. Do some research about the security of DHCP.

CHAPTER

# 19 Domain Name System (DNS)

In this chapter, we discuss the second application program, Domain Name System (DNS). DNS is a client/server application program used to help other application programs. DNS is used to map a host name in the application layer to an IP address in the network layer.

## OBJECTIVES

*The chapter has several objectives:*

- To describe the purpose of DNS.
- To define the concept of domains and domain name space.
- To describe the distribution of name spaces and define zones.
- To discuss the use of DNS in the Internet and describe three categories of domains: generic, country, and reverse.
- To discuss name-address resolution and show the two resolution methods: recursive and iterative.
- To show the format of DNS message and how they can be compressed.
- To discuss DDNS and DNSSEC.

## 19.1 NEED FOR DNS

To identify an entity, TCP/IP protocols use the IP address, which uniquely identifies the connection of a host to the Internet. However, people prefer to use names instead of numeric addresses. Therefore, we need a system that can map a name to an address or an address to a name.

When the Internet was small, mapping was done using a *host file*. The host file had only two columns: name and address. Every host could store the host file on its disk and update it periodically from a master host file. When a program or a user wanted to map a name to an address, the host consulted the host file and found the mapping.

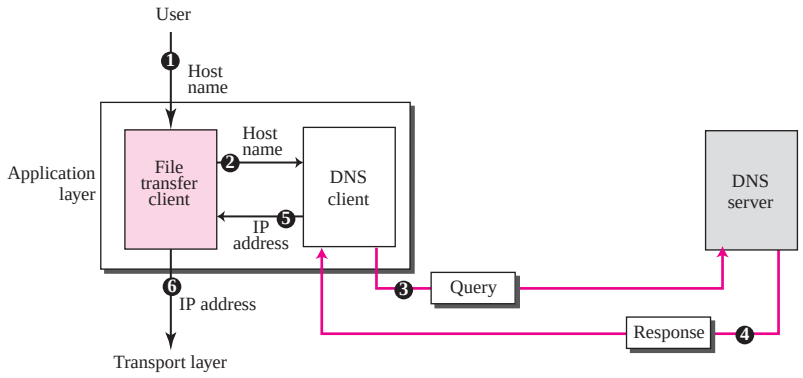
Today, however, it is impossible to have one single host file to relate every address with a name and vice versa. The host file would be too large to store in every host. In addition, it would be impossible to update all the host files every time there is a change.

One solution would be to store the entire host file in a single computer and allow access to this centralized information to every computer that needs mapping. But we know that this would create a huge amount of traffic on the Internet.

Another solution, the one used today, is to divide this huge amount of information into smaller parts and store each part on a different computer. In this method, the host that needs mapping can contact the closest computer holding the needed information. This method is used by the **Domain Name System (DNS).** In this chapter, we first discuss the concepts and ideas behind the DNS. We then describe the DNS protocol itself.

Figure 19.1 shows how TCP/IP uses a DNS client and a DNS server to map a name to an address; the reverse mapping is similar.

**Figure 19.1** *Purpose of DNS*



In Figure 19.1, a user wants to use a file transfer client to access the corresponding file transfer server running on a remote host. The user knows only the file transfer server name, such as *forouzan.com*. However, the TCP/IP suite needs the IP address of the file transfer server to make the connection. The following six steps map the host name to an IP address.

1. The user passes the host name to the file transfer client.
2. The file transfer client passes the host name to the DNS client.
3. We know from Chapter 18 that each computer, after being booted, knows the address of one DNS server. The DNS client sends a message to a DNS server with a query that gives the file transfer server name using the known IP address of the DNS server.
4. The DNS server responds with the IP address of the desired file transfer server.
5. The DNS client passes the IP address to the file transfer server.
6. The file transfer client now uses the received IP address to access the file transfer server.

Note that the purpose of accessing the Internet is to make a connection between the file transfer client and server, but before this can happen, another connection needs to be made between the DNS client and DNS server. In other words, we need two connections; the first is for mapping the name to an IP address; the second is for transferring files (for example).

## 19.2 NAME SPACE

To be unambiguous, the names assigned to machines must be carefully selected from a name space with complete control over the binding between the names and IP addresses. In other words, the names must be unique because the addresses are unique. A **name space** that maps each address to a unique name can be organized in two ways: flat or hierarchical.

### Flat Name Space

In a **flat name space,** a name is assigned to an address. A name in this space is a sequence of characters without structure. The names may or may not have a common section; if they do, it has no meaning. The main disadvantage of a flat name space is that it cannot be used in a large system such as the Internet because it must be centrally controlled to avoid ambiguity and duplication.

### Hierarchical Name Space

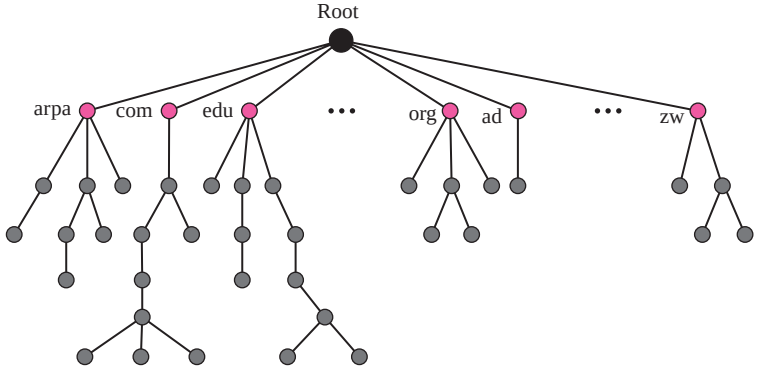
In a **hierarchical name space,** each name is made of several parts. The first part can define the nature of the organization, the second part can define the name of an organization, the third part can define departments in the organization, and so on. In this case, the authority to assign and control the name spaces can be decentralized. A central authority can assign the part of the name that defines the nature of the organization and the name of the organization. The responsibility of the rest of the name can be given to

the organization itself. The organization can add suffixes (or prefixes) to the name to define its host or resources. The management of the organization need not worry that the prefix chosen for a host is taken by another organization because, even if part of an address is the same, the whole address is different. For example, assume two colleges and a company call one of their computers *challenger*. The first college is given a name by the central authority such as *fhda.edu*, the second college is given the name *berkeley.edu*, and the company is given the name *smart.com*. When each of these organizations adds the name *challenger* to the name they have already been given, the end result is three distinguishable names: *challenger.fhda.edu*, *challenger.berkeley.edu*, and *challenger.smart.com*. The names are unique without the need for assignment by a central authority. The central authority controls only part of the name, not the whole.

## Domain Name Space

To have a hierarchical name space, a **domain name space** was designed. In this design the names are defined in an inverted-tree structure with the root at the top. The tree can have only 128 levels: level 0 (root) to level 127 (see Figure 19.2).

**Figure 19.2** *Domain name space*



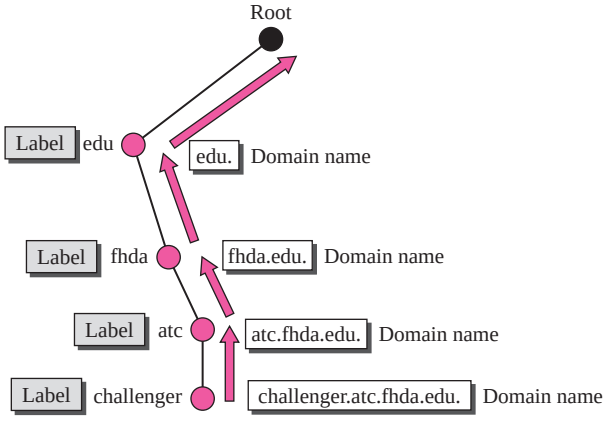

### *Label*

Each node in the tree has a **label,** which is a string with a maximum of 63 characters. The root label is a null string (empty string). DNS requires that children of a node (nodes that branch from the same node) have different labels, which guarantees the uniqueness of the domain names.

### *Domain Name*

Each node in the tree has a domain name. A full **domain name** is a sequence of labels separated by dots (.). The domain names are always read from the node up to the root. The last label is the label of the root (null). This means that a full domain name always ends in a null label, which means the last character is a dot because the null string is nothing. Figure 19.3 shows some domain names.

**Figure 19.3** *Domain names and labels*




**Fully Qualified Domain Name (FQDN)** If a label is terminated by a null string, it is called a **fully qualified domain name (FQDN).** An FQDN is a domain name that contains the full name of a host. It contains all labels, from the most specific to the most general, that uniquely define the name of the host. For example, the domain name is the FQDN of a computer named *challenger* installed at the Advanced Technology Center (ATC) at De Anza College. A DNS server can only match an FQDN to an address. Note that the name must end with a null label, but because null means nothing, the label ends with a dot (.).

**challenger.atc.fhda.edu.**

**Partially Qualified Domain Name (PQDN)** If a label is not terminated by a null string, it is called a **partially qualified domain name (PQDN).** A PQDN starts from a node, but it does not reach the root. It is used when the name to be resolved belongs to the same site as the client. Here the resolver can supply the missing part, called the *suffix*, to create an FQDN. For example, if a user at the *fhda.edu.* site wants to get the IP address of the challenger computer, he or she can define the partial name

**challenger**

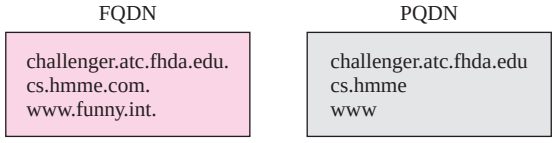
The DNS client adds the suffix *atc.fhda.edu.* before passing the address to the DNS server.

The DNS client normally holds a list of suffixes. The following can be the list of suffixes at De Anza College. The null suffix defines nothing. This suffix is added when the user defines an FQDN.

**atc.fhda.edu fhda.edu null**

Figure 19.4 shows some FQDNs and PQDNs.

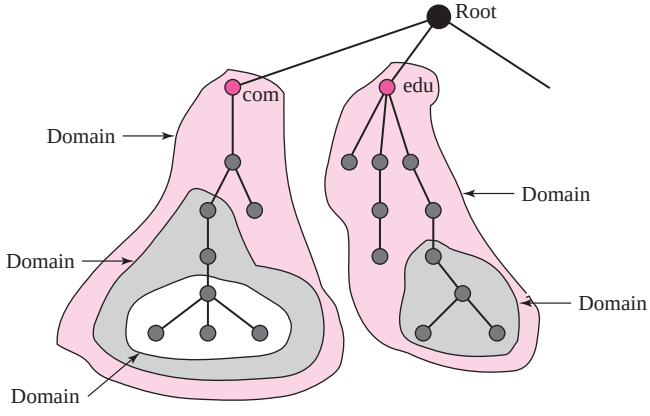
**Figure 19.4** *FQDN and PQDN*




## Domain

A **domain** is a subtree of the domain name space. The name of the domain is the name of the node at the top of the subtree. Figure 19.5 shows some domains. Note that a domain may itself be divided into domains (or **subdomains** as they are sometimes called).

**Figure 19.5** *Domains*




## Distribution of Name Space

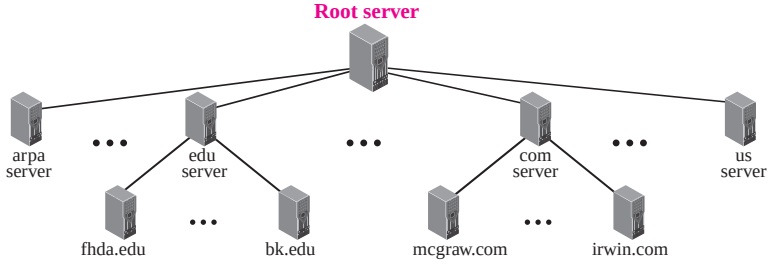
The information contained in the domain name space must be stored. However, it is very inefficient and also not reliable to have just one computer store such a huge amount of information. It is inefficient because responding to requests from all over the world places a heavy load on the system. It is not reliable because any failure makes the data inaccessible.

### *Hierarchy of Name Servers*

The solution to these problems is to distribute the information among many computers called **DNS servers.** One way to do this is to divide the whole space into many domains based on the first level. In other words, we let the root stand alone and create as many

domains (subtrees) as there are first-level nodes. Because a domain created this way could be very large, DNS allows domains to be divided further into smaller domains (subdomains). Each server can be responsible (authoritative) for either a large or small domain. In other words, we have a hierarchy of servers in the same way that we have a hierarchy of names (see Figure 19.6).

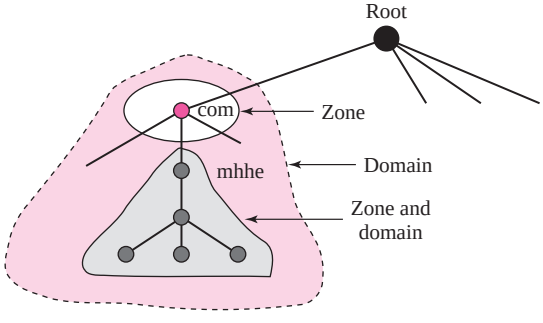
**Figure 19.6** *Hierarchy of name servers*




### *Zone*

Since the complete domain name hierarchy cannot be stored on a single server, it is divided among many servers. What a server is responsible for or has authority over is called a **zone.** We can define a zone as a contiguous part of the entire tree. If a server accepts responsibility for a domain and does not divide the domain into smaller domains, the "domain" and the "zone" refer to the same thing. The server makes a database called a *zone file* and keeps all the information for every node under that domain. However, if a server divides its domain into subdomains and delegates part of its authority to other servers, "domain" and "zone" refer to different things. The information about the nodes in the subdomains is stored in the servers at the lower levels, with the original server keeping some sort of reference to these lower-level servers. Of course the original server does not free itself from responsibility totally: It still has a zone, but the detailed information is kept by the lower-level servers (see Figure 19.7).

**Figure 19.7** *Zones and domains*



A server can also divide part of its domain and delegate responsibility but still keep part of the domain for itself. In this case, its zone is made of detailed information for the part of the domain that is not delegated and references to those parts that are delegated.

### *Root Server*

A **root server** is a server whose zone consists of the whole tree. A root server usually does not store any information about domains but delegates its authority to other servers, keeping references to those servers. There are several root servers, each covering the whole domain name space. The root servers are distributed all around the world.

### *Primary and Secondary Servers*

DNS defines two types of servers: primary and secondary. A **primary server** is a server that stores a file about the zone for which it is an authority. It is responsible for creating, maintaining, and updating the zone file. It stores the zone file on a local disk.

A **secondary server** is a server that transfers the complete information about a zone from another server (primary or secondary) and stores the file on its local disk. The secondary server neither creates nor updates the zone files. If updating is required, it must be done by the primary server, which sends the updated version to the secondary.

The primary and secondary servers are both authoritative for the zones they serve. The idea is not to put the secondary server at a lower level of authority but to create redundancy for the data so that if one server fails, the other can continue serving clients. Note also that a server can be a primary server for a specific zone and a secondary server for another zone. Therefore, when we refer to a server as a primary or secondary server, we should be careful about which zone we refer to.

**A primary server loads all information from the disk file; the secondary server loads all information from the primary server. When the secondary downloads information from the primary, it is called zone transfer.**

## 19.3 DNS IN THE INTERNET

DNS is a protocol that can be used in different platforms. In the Internet, the domain name space (tree) is divided into three different sections: generic domains, country domains, and the inverse domain (see Figure 19.8).

### Generic Domains

The **generic domains** define registered hosts according to their generic behavior. Each node in the tree defines a domain, which is an index to the domain name space database (see Figure 19.9).

**Figure 19.8** *DNS used in the Internet*

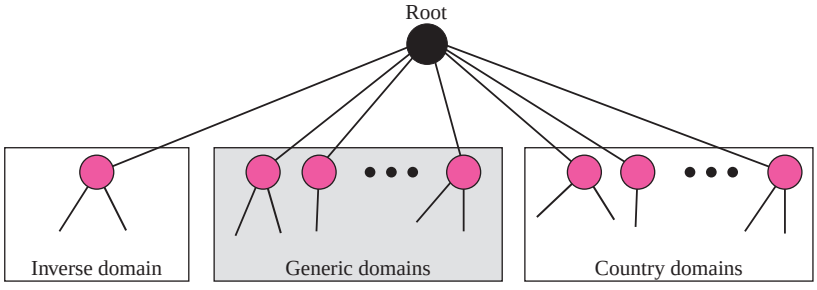
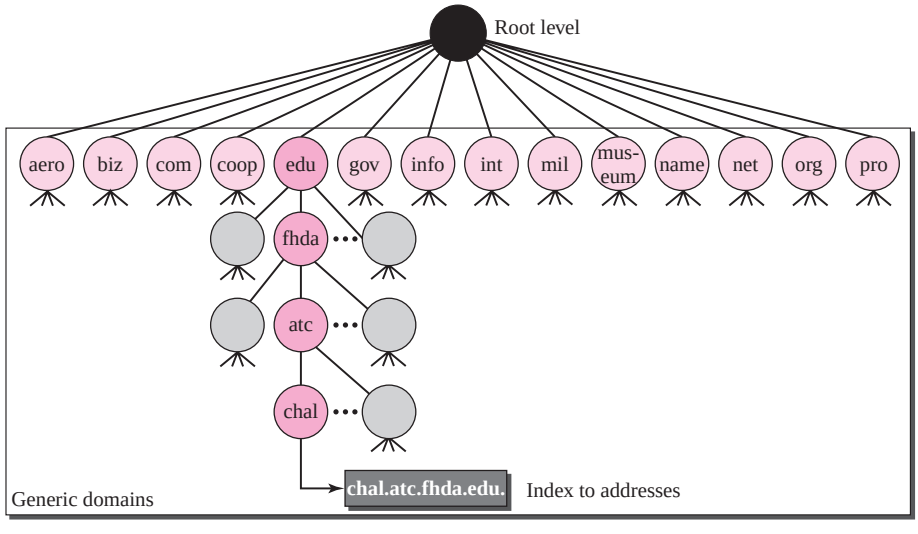



**Figure 19.9** *Generic domains*




Looking at the tree, we see that the first level in the generic domains section allows 14 possible labels. These labels describe the organization types as listed in Table 19.1.

## Country Domains

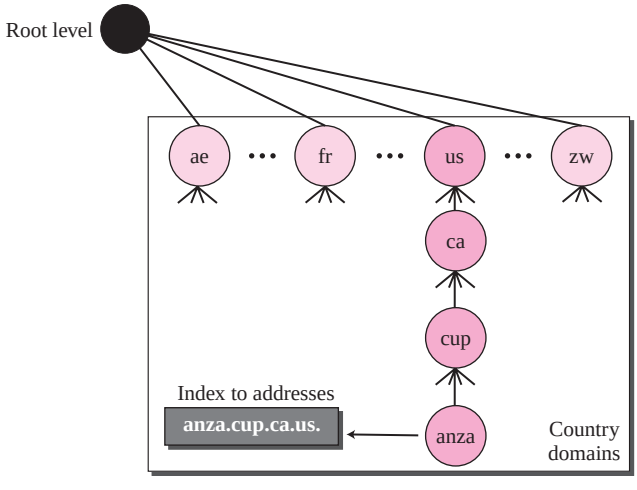
The **country domains** section uses two-character country abbreviations (e.g., us for United States). Second labels can be organizational, or they can be more specific, national designations. The United States, for example, uses state abbreviations as a subdivision of us (e.g., ca.us.).

Figure 19.10 shows the country domains section. The address *anza.cup.ca.us* can be translated to De Anza College in Cupertino in California in the United States.

**Table 19.1** *Generic domain labels*

| *Label* | *Description* |
|---|---|
| **aero** | Airlines and aerospace companies |
| **biz** | Businesses or firms (similar to "com") |
| **com** | Commercial organizations |
| **coop** | Cooperative business organizations |
| **edu** | Educational institutions |
| **gov** | Government institutions |
| **info** | Information service providers |
| **int** | International organizations |
| **mil** | Military groups |
| **museum** | Museums and other non-profit organizations |
| **name** | Personal names (individuals) |
| **net** | Network support centers |
| **org** | Nonprofit organizations |
| **pro** | Professional individual organizations |

**Figure 19.10** *Country domains*




## Inverse Domain

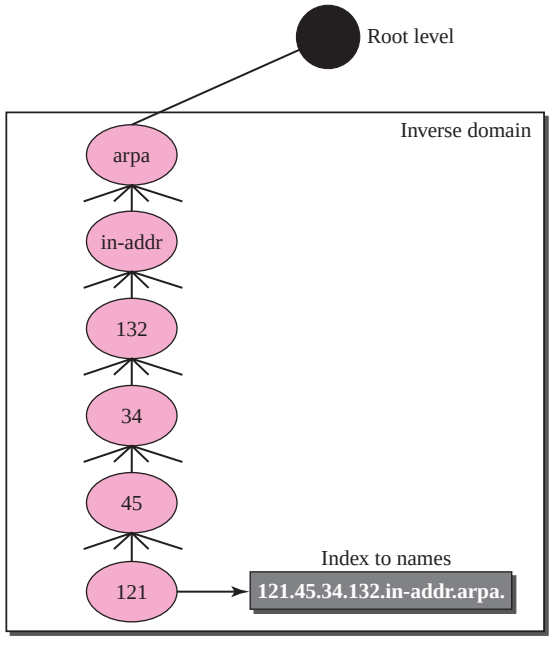
The **inverse domain** is used to map an address to a name. This may happen, for example, when a server has received a request from a client to do a task. Although the server has a file that contains a list of authorized clients, only the IP address of the client (extracted from the received IP packet) is listed. The server asks its resolver to send a

query to the DNS server to map an address to a name to determine if the client is on the authorized list.

This type of query is called an inverse or pointer (PTR) query. To handle a pointer query, the inverse domain is added to the domain name space with the first-level node called *arpa* (for historical reasons). The second level is also one single node named *in-addr* (for inverse address). The rest of the domain defines IP addresses.

The servers that handle the inverse domain are also hierarchical. This means the netid part of the address should be at a higher level than the subnetid part, and the subnetid part higher than the hostid part. In this way, a server serving the whole site is at a higher level than the servers serving each subnet. This configuration makes the domain look inverted when compared to a generic or country domain. To follow the convention of reading the domain labels from the bottom to the top, an IP address such as 132.34.45.121 (a class B address with netid 132.34) is read as 121.45.34.132.in-addr.arpa. See Figure 19.11 for an illustration of the inverse domain configuration.

**Figure 19.11** *Inverse domain*




## Registrar

How are the new domains added to DNS? This is done through a **registrar,** a commercial entity accredited by ICANN. A registrar first verifies that the requested domain name is unique and then enters it into the DNS database. A fee is charged.

## 19.4 RESOLUTION

Mapping a name to an address or an address to a name is called *name-address resolution.*

### Resolver

DNS is designed as a client-server application. A host that needs to map an address to a name or a name to an address calls a DNS client called a **resolver.** The resolver accesses the closest DNS server with a mapping request. If the server has the information, it satisfies the resolver; otherwise, it either refers the resolver to other servers or asks other servers to provide the information.

After the resolver receives the mapping, it interprets the response to see if it is a real resolution or an error, and finally delivers the result to the process that requested it.

### Mapping Names to Addresses

Most of the time, the resolver gives a domain name to the server and asks for the corresponding address. In this case, the server checks the generic domains or the country domains to find the mapping.

If the domain name is from the generic domains section, the resolver receives a domain name such as "*chal.atc.fhda.edu.*". The query is sent by the resolver to the local DNS server for resolution. If the local server cannot resolve the query, it either refers the resolver to other servers or asks other servers directly.

If the domain name is from the country domains section, the resolver receives a domain name such as "*ch.fhda.cu.ca.us.*". The procedure is the same.

### Mapping Addresses to Names

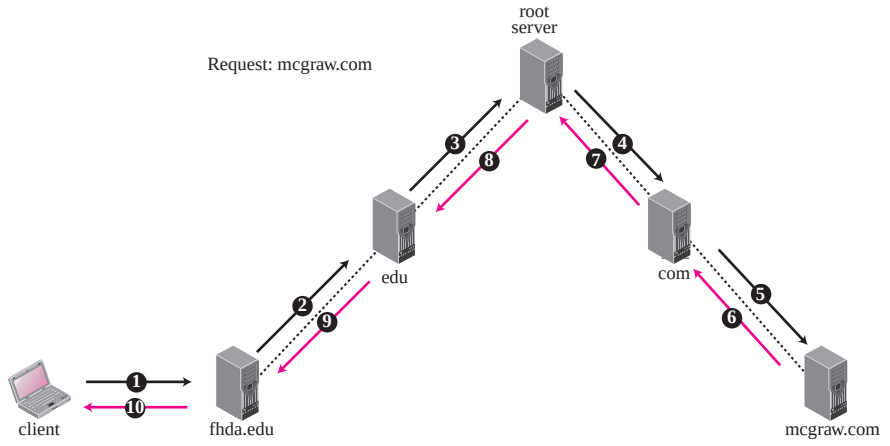
A client can send an IP address to a server to be mapped to a domain name. As mentioned before, this is called a PTR query. To answer queries of this kind, DNS uses the inverse domain. However, in the request, the IP address is reversed and two labels, *in-addr* and *arpa,* are appended to create a domain acceptable by the inverse domain section. For example, if the resolver receives the IP address 132.34.45.121, the resolver first inverts the address and then adds the two labels before sending. The domain name sent is "*121.45.34.132.in-addr.arpa*.", which is received by the local DNS and resolved.

### Recursive Resolution

Figure 19.12 shows the recursive resolution.

The client (resolver) can ask for a recursive answer from a name server. This means that the resolver expects the server to supply the final answer. If the server is the authority for the domain name, it checks its database and responds. If the server is not the authority, it sends the request to another server (the parent usually) and waits for the response. If the parent is the authority, it responds; otherwise, it sends the query to yet another server. When the query is finally resolved, the response travels back until it finally reaches the requesting client.

**Figure 19.12** *Recursive resolution*




## Iterative Resolution

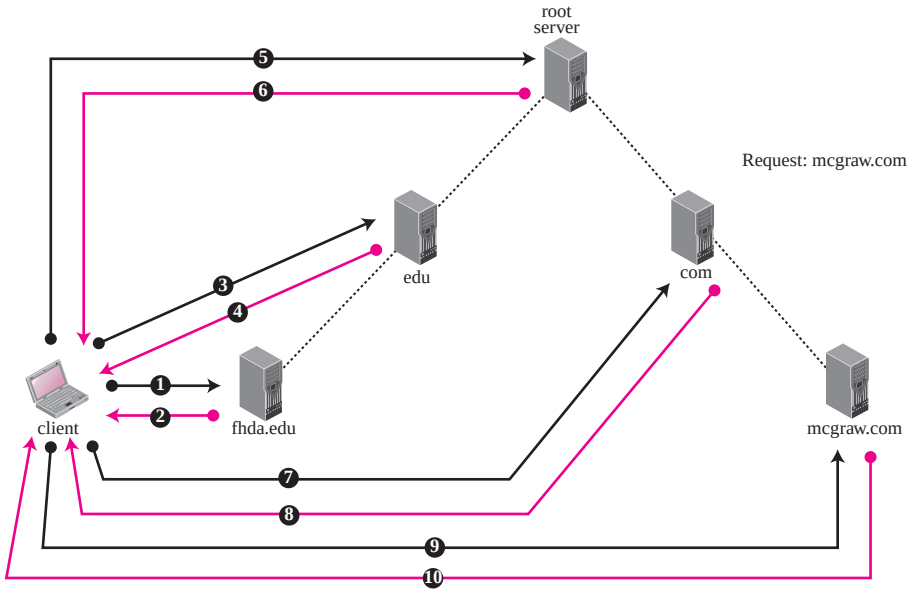
If the client does not ask for a recursive answer, the mapping can be done iteratively. If the server is an authority for the name, it sends the answer. If it is not, it returns (to the client) the IP address of the server that it thinks can resolve the query. The client is responsible for repeating the query to this second server. If the newly addressed server can resolve the problem, it answers the query with the IP address; otherwise, it returns the IP address of a new server to the client. Now the client must repeat the query to the third server. This process is called *iterative* because the client repeats the same query to multiple servers. In Figure 19.13 the client queries five servers before it gets an answer from the mcgraw.com server.

## Caching

Each time a server receives a query for a name that is not in its domain, it needs to search its database for a server IP address. Reduction of this search time would increase efficiency. DNS handles this with a mechanism called **caching.** When a server asks for a mapping from another server and receives the response, it stores this information in its cache memory before sending it to the client. If the same or another client asks for the same mapping, it can check its cache memory and resolve the problem. However, to inform the client that the response is coming from the cache memory and not from an authoritative source, the server marks the response as *unauthoritative.*

Caching speeds up resolution, but it can also be problematic. If a server caches a mapping for a long time, it may send an outdated mapping to the client. To counter this, two techniques are used. First, the authoritative server always adds information to the mapping called *time-to-live* (TTL). It defines the time in seconds that the receiving server can cache the information. After that time, the mapping is invalid and any query must be sent again to the authoritative server. Second, DNS requires

**Figure 19.13** *Iterative resolution*

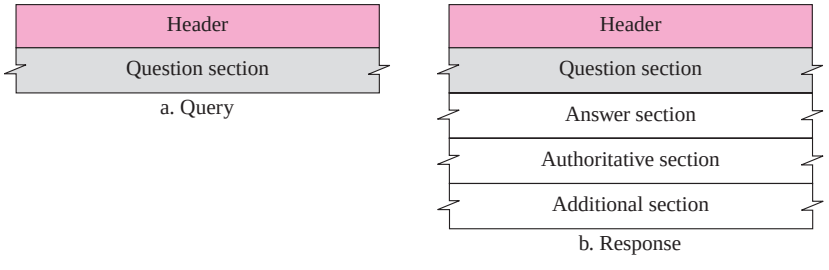



that each server keep a TTL counter for each mapping it caches. The cache memory must be searched periodically and those mappings with an expired TTL must be purged.

## 19.5 DNS MESSAGES

DNS has two types of messages: query and response. Both types have the same format. The query message consists of a header and question records; the response message consists of a header, question records, answer records, authoritative records, and additional records (see Figure 19.14).

**Figure 19.14** *Query and response messages*



## Header

Both query and response messages have the same header format with some fields set to zero for the query messages. The header is 12 bytes and its format is shown in Figure 19.15.

**Figure 19.15** *Header format*

| Identification | Flags |
|---|---|
| Number of question records | Number of answer records (All 0s in query message) |
| Number of authoritative records (All 0s in query message) | Number of additional records (All 0s in query message) |

The header fields are as follows:

- **Identification.** This is a 16-bit field used by the client to match the response with the query. The client uses a different identification number each time it sends a query. The server duplicates this number in the corresponding response.
- **Flags.** This is a 16-bit field consisting of the subfields shown in Figure 19.16.

**Figure 19.16** *Flags field*


A brief description of each flag subfield follows.

a. **QR (query/response).** This is a 1-bit subfield that defines the type of message. If it is 0, the message is a query. If it is 1, the message is a response.

b. **OpCode.** This is a 4-bit subfield that defines the type of query or response (0 if standard, 1 if inverse, and 2 if a server status request).

c. **AA (authoritative answer).** This is a 1-bit subfield. When it is set (value of 1) it means that the name server is an authoritative server. It is used only in a response message.

d. **TC (truncated).** This is a 1-bit subfield. When it is set (value of 1), it means that the response was more than 512 bytes and truncated to 512. It is used when DNS uses the services of UDP (see Section 19.8 on Encapsulation).

e. **RD (recursion desired).** This is a 1-bit subfield. When it is set (value of 1) it means the client desires a recursive answer. It is set in the query message and repeated in the response message.

f. **RA (recursion available).** This is a 1-bit subfield. When it is set in the response, it means that a recursive response is available. It is set only in the response message.

g. **Reserved.** This is a 3-bit subfield set to 000.

h. **rCode.** This is a 4-bit field that shows the status of the error in the response. Of course, only an authoritative server can make such a judgment. Table 19.2 shows the possible values for this field.

**Table 19.2** *Values of rCode*

| *Value* | *Meaning* | *Value* | *Meaning* |
|---|---|---|---|
| 0 | No error | 4 | Query type not supported |
| 1 | Format error | 5 | Administratively prohibited |
| 2 | Problem at name server | 6–15 | Reserved |
| 3 | Domain reference problem | | |

- **Number of question records.** This is a 16-bit field containing the number of queries in the question section of the message.
- **Number of answer records.** This is a 16-bit field containing the number of answer records in the answer section of the response message. Its value is zero in the query message.
- **Number of authoritative records.** This is a 16-bit field containing the number of authoritative records in the authoritative section of a response message. Its value is zero in the query message.
- **Number of additional records.** This is a 16-bit field containing the number of additional records in the additional section of a response message. Its value is zero in the query message.

### *Question Section*

This is a section consisting of one or more question records. It is present on both query and response messages. We will discuss the question records in a following section.

### *Answer Section*

This is a section consisting of one or more resource records. It is present only on response messages. This section includes the answer from the server to the client (resolver). We will discuss resource records in a following section.

### *Authoritative Section*

This is a section consisting of one or more resource records. It is present only on response messages. This section gives information (domain name) about one or more authoritative servers for the query.

### *Additional Information Section*

This is a section consisting of one or more resource records. It is present only on response messages. This section provides additional information that may help the resolver. For example, a server may give the domain name of an authoritative server to the resolver in the authoritative section, and include the IP address of the same authoritative server in the additional information section.

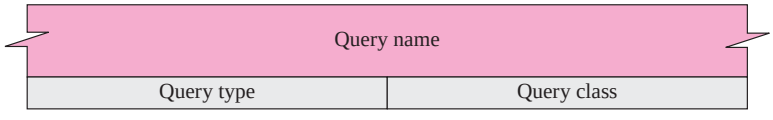
## 19.6 TYPES OF RECORDS

As we saw in the previous section, two types of records are used in DNS. The question records are used in the question section of the query and response messages. The resource records are used in the answer, authoritative, and additional information sections of the response message.

### Question Record

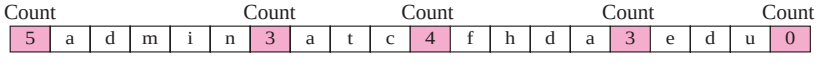
A **question record** is used by the client to get information from a server. This contains the domain name. Figure 19.17 shows the format of a question record. The list below describes question record fields.

**Figure 19.17** *Question record format*




- **Query name.** This is a variable-length field containing a domain name (see Figure 19.18). The count field refers to the number of characters in each section.

**Figure 19.18** *Query name format*




- **Query type.** This is a 16-bit field defining the type of query. Table 19.3 shows some of the types commonly used. The last two can only be used in a query.

**Table 19.3** *Types*

| *Type* | *Mnemonic* | *Description* |
|---|---|---|
| 1 | A | **Address.** A 32-bit IPv4 address. It converts a domain name to an address. |
| 2 | NS | **Name server**. It identifies the authoritative servers for a zone. |
| 5 | CNAME | **Canonical name**. It defines an alias for the official name of a host. |
| 6 | SOA | **Start of authority**. It marks the beginning of a zone. |
| 11 | WKS | **Well-known services**. It defines the network services that a host provides. |
| 12 | PTR | **Pointer.** It is used to convert an IP address to a domain name. |
| 13 | HINFO | **Host information.** It defines the hardware and operating system. |
| 15 | MX | **Mail exchange**. It redirects mail to a mail server. |
| 28 | AAAA | **Address.** An IPv6 address (see Chapter 26). |
| 252 | AXFR | A request for the transfer of the entire zone. |
| 255 | ANY | A request for all records. |

- **Query class.** This is a 16-bit field defining the specific protocol using DNS. Table 19.4 shows the current values. In this text we are interested only in class 1 (the Internet).

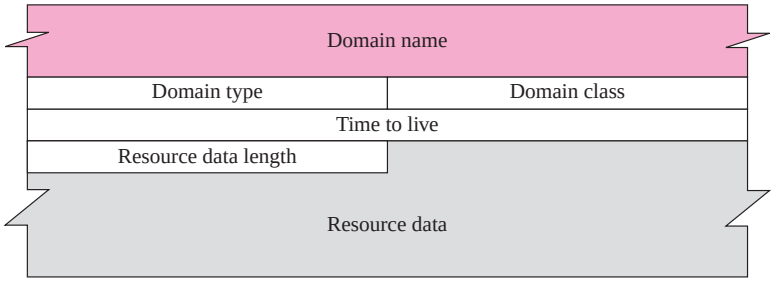
**Table 19.4** *Classes*

| *Class* | *Mnemonic* | *Description* |
|---|---|---|
| 1 | IN | Internet |
| 2 | CSNET | CSNET network (obsolete) |
| 3 | CS | The COAS network |
| 4 | HS | The Hesiod server developed by MIT |

## Resource Record

Each domain name (each node on the tree) is associated with a record called the **resource record.** The server database consists of resource records. Resource records are also what is returned by the server to the client. Figure 19.19 shows the format of a resource record.

**Figure 19.19** *Resource record format*



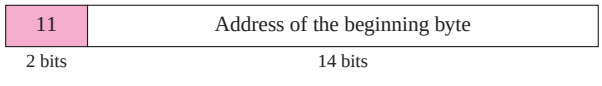

- **Domain name.** This is a variable-length field containing the domain name. It is a duplication of the domain name in the question record. Since DNS requires the use of compression everywhere a name is repeated, this field is a pointer offset to the corresponding domain name field in the question record. See Section 19.7 on Compression.
- **Domain type.** This field is the same as the query type field in the question record except the last two types are not allowed. Refer to Table 19.3 for more information.
- **Domain class.** This field is the same as the query class field in the question record (see Table 19.4).
- **Time-to-live.** This is a 32-bit field that defines the number of seconds the answer is valid. The receiver can cache the answer for this period of time. A zero value means that the resource record is used only in a single transaction and is not cached.
- **Resource data length.** This is a 16-bit field defining the length of the resource data.

- **Resource data.** This is a variable-length field containing the answer to the query (in the answer section) or the domain name of the authoritative server (in the authoritative section) or additional information (in the additional information section). The format and contents of this field depend on the value of the type field. It can be one of the following:
  - **a. A number.** This is written in octets. For example, an IPv4 address is a 4-octet integer and an IPv6 address is a 16-octet integer.
  - **b. A domain name.** Domain names are expressed as a sequence of labels. Each label is preceded by a 1-byte length field that defines the number of characters in the label. Since every domain name ends with the null label, the last byte of every domain name is the length field with the value of 0. To distinguish between a length field and an offset pointer (as we will discuss later), the two high-order bits of a length field are always zero (00). This will not create a problem because the length of a label cannot be more than 63, which is a maximum of 6 bits (111111).
  - **c. An offset pointer.** Domain names can be replaced with an offset pointer. An offset pointer is a 2-byte field with each of the 2 high-order bits set to 1 (11).
  - **d. A character string.** A character string is represented by a 1-byte length field followed by the number of characters defined in the length field. The length field is not restricted like the domain name length field. The character string can be as long as 255 characters (including the length field).

## 19.7 COMPRESSION

DNS requires that a domain name be replaced by an offset pointer if it is repeated. For example, in a resource record the domain name is usually a repetition of the domain name in the question record. For efficiency, DNS defines a 2-byte offset pointer that points to a previous occurrence of the domain or part of it. The format of the field is shown in Figure 19.20.

**Figure 19.20** *Format of an offset pointer*

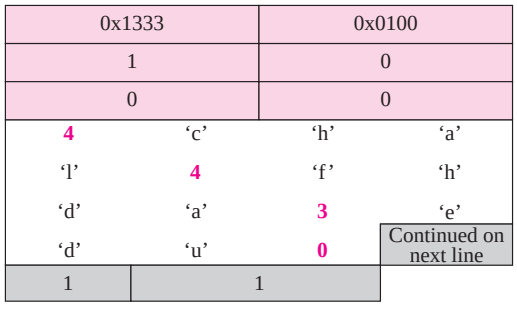



The first 2 high-order bits are two 1s to distinguish an offset pointer from a length field. The other 14 bits represent a number that points to the corresponding byte number in the message. The bytes in a message are counted from the beginning of the message with the first byte counted as byte 0. For example, if an offset pointer refers to byte 12 (the 13th byte) of the message, the value should be 1100000000001100. Here the 2 leftmost bits define the field as an offset pointer and the other bits define the decimal number 12. We will show the use of the offset pointers in the following examples.

### Example 19.1

A resolver sends a query message to a local server to find the IP address for the host "*chal.fhda.edu.*". We discuss the query and response messages separately. Figure 19.21 shows the query message sent by the resolver.

**Figure 19.21** *Example 19.1: Query message*



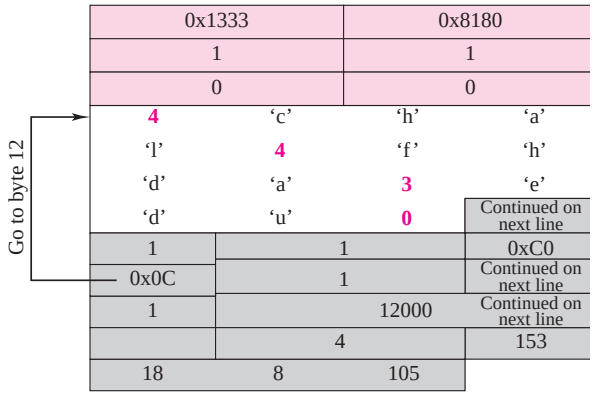

The first 2 bytes show the identifier $(1333)_{16}$. It is used as a sequence number and relates a response to a query. Because a resolver may even send many queries to the same server, the identifier helps to sort responses that arrive out of order. The next bytes contain the flags with the value of 0x0100 in hexadecimal. In binary it is 0000000100000000, but it is more meaningful to divide it into the fields as shown below:

| QR | OpCode | AA | TC | RD | RA | Reserved | rCode |
|---|---|---|---|---|---|---|---|
| 0 | 0000 | 0 | 0 | 1 | 0 | 000 | 0000 |

The QR bit defines the message as a query. The OpCode is 0000, which defines a standard query. The recursion desired (RD) bit is set. (Refer back to Figure 19.16 for the flags field descriptions.) The message contains only one question record. The domain name is ***4****chal****4****fhda****3****edu****0***. The next 2 bytes define the query type as an IP address; the last 2 bytes define the class as the Internet.

Figure 19.22 shows the response of the server.

**Figure 19.22** *Example 19.1: Response message*



The response is similar to the query except that the flags are different and the number of answer records is one. The flags value is 0x8180 in hexadecimal. In binary it is 1000000110000000, but again we divide it into fields as shown below:

| QR | OpCode | AA | TC | RD | RA | Reserved | rCode |
|---|---|---|---|---|---|---|---|
| 1 | 0000 | 0 | 0 | 1 | 1 | 000 | 0000 |

The QR bit defines the message as a response. The OpCode is 0000, which defines a standard response. The recursion available (RA) and RD bits are set. The message contains one question record and one answer record. The question record is repeated from the query message. The answer record has a value of 0xC00C (split in two lines), which points to the question record instead of repeating the domain name. The next field defines the domain type (address). The field after that defines the class (Internet). The field with the value 12,000 is the TTL (12,000 s). The next field is the length of the resource data, which is an IP address (153.18.8.105).

## Example 19.2

An FTP server has received a packet from an FTP client with IP address 153.2.7.9. The FTP server wants to verify that the FTP client is an authorized client. The FTP server can consult a file containing the list of authorized clients. However, the file consists only of domain names. The FTP server has only the IP address of the requesting client, which was the source IP address in the received IP datagram. The FTP server asks the resolver (DNS client) to send an inverse query to a DNS server to ask for the name of the FTP client. We discuss the query and response messages separately. Figure 19.23 shows the query message sent from the resolver to the server.

**Figure 19.23** *Example 19.2: Inverse query message*

| 0x1200 | | 0x0900 | |
|---|---|---|---|
| 1 | | 0 | |
| 0 | | 0 | |
| **1** | '9' | **1** | '7' |
| **1** | '2' | **3** | 'l' |
| '5' | '3' | **7** | 'i' |
| 'n' | '-' | 'a' | 'd' |
| 'd' | 'r' | **4** | 'a' |
| 'r' | 'p' | 'a' | **0** |
| 12 | | 1 | |

The first 2 bytes show the identifier (0x1200). The flags value is 0x0900 in hexadecimal. In binary it is 0000100100000000, and we divide it into fields as shown below:

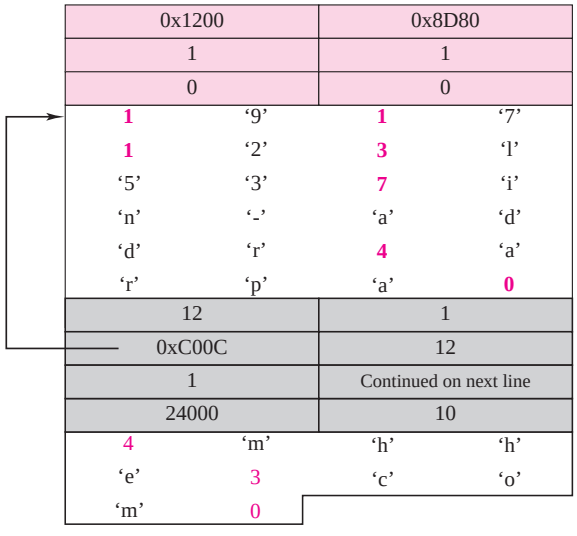
| QR | OpCode | AA | TC | RD | RA | Reserved | rCode |
|---|---|---|---|---|---|---|---|
| 0 | 0001 | 0 | 0 | 1 | 0 | 000 | 0000 |

The OpCode is 0001, which defines an inverse query. The message contains only one question record. The domain name is *19171231537in-addr4arpa*. The next 2 bytes define the query type as PTR, and the last 2 bytes define the class as the Internet.

Figure 19.24 shows the response. The flags value is 0x8D80 in hexadecimal. In binary it is 1000110110000000, and we divide it into fields as shown below:

| QR | OpCode | AA | TC | RD | RA | Reserved | rCode |
|---|---|---|---|---|---|---|---|
| 1 | 0001 | 1 | 0 | 1 | 0 | 000 | 0000 |

**Figure 19.24** *Example 19.2: Inverse response message*




The message contains one question record and one answer record. The question record is repeated from the query message. The answer record has a value of 0xC00C, which points to the question record instead of repeating the domain name. The next field defines the domain type (PTR). The field after that defines the class (Internet), and the field after that defines the TTL (24,000 s). The next field is the length of the resource data (10). The last field is the domain name *4mhhe3com0*, which means "mhhe.com.".

## Example 19.3

In UNIX and Windows, the *nslookup* utility can be used to retrieve address/name mapping. The following shows how we can retrieve an address when the domain name is given.

```
$ nslookup  fhda.edu
Name:   fhda.edu
Address: 153.18.8.1
```

The *nslookup* utility can also be used to retrieve the domain name when the address is given as shown below:

```
$ nslookup 153.18.8.1
1.8.18.153.in-addr.arpa name = tiptoe.fhda.edu.
```

## 19.8 ENCAPSULATION

DNS can use either UDP or TCP. In both cases the well-known port used by the server is port 53. UDP is used when the size of the response message is less than 512 bytes because most UDP packages have a 512-byte packet size limit. If the size of the response message is more than 512 bytes, a TCP connection is used. In that case, one of two scenarios can occur:

- If the resolver has prior knowledge that the size of the response message is more than 512 bytes, it uses the TCP connection. For example, if a secondary name server (acting as a client) needs a zone transfer from a primary server, it uses the TCP connection because the size of the information being transferred usually exceeds 512 bytes.
- If the resolver does not know the size of the response message, it can use the UDP port. However, if the size of the response message is more than 512 bytes, the server truncates the message and turns on the TC bit (See Figure 19.16). The resolver now opens a TCP connection and repeats the request to get a full response from the server.

## 19.9 REGISTRARS

How are new domains added to DNS? This is done through a **registrar,** a commercial entity accredited by ICANN. A registrar first verifies that the requested domain name is unique and then enters it into the DNS database. A fee is charged. Today, there are many registrars; their names and addresses can be found at

http://www.intenic.net

To register, the organization needs to give the name of its server and the IP address of the server. For example, a new commercial organization named *wonderful* with a server named *ws* and IP address 200.200.200.5 needs to give the following information to one of the registrars:

**Domain name:** ws.wonderful.com
**IP address:** 200.200.200.5

## 19.10 DDNS

When the DNS was designed, no one predicted that there would be so many address changes. In DNS, when there is a change, such as adding a new host, removing a host, or changing an IP address, the change must be made to the DNS master file. These

types of changes involve a lot of manual updating. The size of today's Internet does not allow for this kind of manual operation.

The DNS master file must be updated dynamically. The **Dynamic Domain Name System (DDNS)** therefore was devised to respond to this need. In DDNS, when a binding between a name and an address is determined, the information is sent, usually by DHCP (see Chapter 18) to a primary DNS server. The primary server updates the zone. The secondary servers are notified either actively or passively. In active notification, the primary server sends a message to the secondary servers about the change in the zone, whereas in passive notification, the secondary servers periodically check for any changes. In either case, after being notified about the change, the secondary requests information about the entire zone (zone transfer).

To provide security and prevent unauthorized changes in the DNS records, DDNS can use an authentication mechanism.

## 19.11 SECURITY OF DNS

DNS is one of the most important systems in the Internet infrastructure; it provides crucial services to the Internet users. Applications such as Web access or e-mail are heavily dependent on the proper operation of DNS. DNS can be attacked in several ways including:

1. The attacker may read the response of a DNS server to find the nature or names of sites the user mostly accesses. This type of information can be used to find the user's profile. To prevent this attack, DNS message needs to be confidential (See Chapter 29).
2. The attacker may intercept the response of a DNS server and change it or create a totally new bogus response to direct the user to the site or domain the attacker wishes the user to access. This type of attack can be protected using message origin authentication and message integrity (See Chapter 29).
3. The attacker may flood the DNS server to overwhelm it or eventually crash it. This type of attack can be protected using the provision against denial-of-service attack.

To protect DNS, IETF has devised a technology named **DNS Security** (**DNSSEC**) that provides the message origin authentication and message integrity using a security service called *digital signature* (See Chapter 29). DNSSEC however, does not provide confidentiality for the DNS messages. There is no specific protection against the denial-of-service attack in the specification of DNSSEC. However, the caching system protects the upper-level servers against this attack to some extent.

## 19.12 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books and RFCs give an easy but thorough coverage of DNS including [Tan 03], [Ste 94], and [Com 06].

### RFCs

DNS has gone through many changes; several RFCs show different updates on DNS including RFC 1034, FRC 1035, RFC 1996, RFC 2535, RFC 3008, RFC 3658, RFC 3755, RFC 3757, and RFC 3845.

## 19.13 KEY TERMS

caching
compression
country domain
DNS server
domain
domain name
domain name space
Domain Name System (DNS)
Dynamic Domain Name System (DDNS)
flat name space
fully qualified domain name (FQDN)
generic domain
hierarchical name space
inverse domain
iterative resolution
label
name space
partially qualified domain name (PQDN)
primary server
question record
registrar
resolver
resource record
root server
secondary server
subdomain
zone

## 19.14 SUMMARY

- The Domain Name System (DNS) is a client-server application that identifies each host on the Internet with a unique user-friendly name. DNS organizes the name space in a hierarchical structure to decentralize the responsibilities involved in naming.
- DNS can be pictured as an inverted hierarchical tree structure with one root node at the top and a maximum of 128 levels. Each node in the tree has a domain name. A domain is defined as any subtree of the domain name space.
- The name space information is distributed among DNS servers. Each server has jurisdiction over its zone. A root server's zone is the entire DNS tree. A primary server creates, maintains, and updates information about its zone. A secondary server gets its information from a primary server.
- The domain name space is divided into three sections: generic domains, country domains, and inverse domain. There are fourteen generic domains, each specifying an organization type. Each country domain specifies a country. The inverse

domain finds a domain name for a given IP address. This is called address-to-name resolution.

- Name servers, computers that run the DNS server program, are organized in a hierarchy. The DNS client, called a resolver, maps a name to an address or an address to a name. In recursive resolution, the client sends its request to a server that eventually returns a response. In iterative resolution, the client may send its request to multiple servers before getting an answer. Caching is a method whereby an answer to a query is stored in memory (for a limited time) for easy access to future requests.
- A fully qualified domain name (FQDN) is a domain name consisting of labels beginning with the host and going back through each level to the root node. A partially qualified domain name (PQDN) is a domain name that does not include all the levels between the host and the root node.
- There are two types of DNS messages: queries and responses. There are two types of DNS records: question records and resource records. DNS uses an offset pointer for duplicated domain name information in its messages. Dynamic DNS (DDNS) automatically updates the DNS master file. DNS uses the services of UDP for messages of less than 512 bytes; otherwise, TCP is used.
- To protect DNS, IETF has devised a technology named DNS Security (DNSSEC) that provides the message origin authentication and message integrity using a security service called digital signature.

## 19.15 PRACTICE SET

### Exercises

1. Determine which of the following is an FQDN and which is a PQDN:
   - **a.** xxx
   - **b.** xxx.yyy.
   - **c.** xxx.yyy.net
   - **d.** zzz.yyy.xxx.edu.
2. Determine which of the following is an FQDN and which is a PQDN:
   - **a.** mil.
   - **b.** edu.
   - **c.** xxx.yyy.net
   - **d.** zzz.yyy.xxx.edu
3. Find the value of the flags field (in hexadecimal) for a query message requesting an address and demanding a recursive answer.
4. Find the value of the flags field (in hexadecimal) for an unauthoritative message carrying an inverse response. The resolver had asked for a recursive response, but the recursive answer was not available.

5. Analyze the flag 0x8F80.
6. Analyze the flag 0x0503. Is it valid?
7. Is the size of a question record fixed?
8. Is the size of a resource record fixed?
9. What is the size of a question record containing the domain name fhda.edu?
10. What is the size of a question record containing an IP address?
11. What is the size of a resource record containing the domain name fhda.edu?
12. What is the size of a resource record containing an IP address?
13. What is the size of a query message requesting the IP address for challenger.atc.fhda.edu?
14. What is the size of a query message requesting the domain name for 185.34.23.12?
15. What is the size of the response message responding to the query message in Exercise 13?
16. What is the size of the response message responding to the query message in Exercise 14?
17. Redo Example 19.1 using a response message with one answer record and one authoritative record which defines "fhda.edu." as the authoritative server.
18. Redo Exercise 17, but add one additional record that defines the address of the authoritative server as 153.18.9.0.
19. A DNS client is looking for the IP address of xxx.yyy.com. Show the query message with values for each field.
20. Show the response message of a DNS server to Exercise 19. Assume the IP address is 201.34.23.12.
21. A DNS client is looking for the IP addresses corresponding to xxx.yyy.com and aaa.bbb.edu. Show the query message.
22. Show the response message of a DNS server to the query in Exercise 21 if the addresses are 14.23.45.12 and 131.34.67.89.
23. Show the response message of Exercise 22 if the DNS server can resolve the first enquiry but not the second.
24. A DNS client is looking for the name of the computer with IP address 132.1.17.8. Show the query message.
25. Show the response message sent by the server to the query in Exercise 24.
26. Encapsulate the query message of Exercise 24 in a UDP user datagram.
27. Encapsulate the response message of Exercise 25 in a UDP user datagram.

## Research Activities

28. Compare and contrast the DNS structure with the UNIX directory structure.
29. What is the equivalent of dots in the DNS structure for the UNIX directory structure?
30. A DNS domain name starts with a node and goes up to the root of the tree. Do the pathnames in UNIX do the same?

**31.** Can we say that the FQDNs in DNS are the same as absolute pathnames in UNIX and PQDNs are the same as relative pathnames in UNIX?

**32.** Find how to use the *nslookup* utility in Windows.

**33.** Find all the options of the *nslookup* utility.

**34.** Try the *nslookup* utility on some domain name you are familiar with.

**35.** Use the *nslookup* utility to find the address of some commercial web servers.

CHAPTER

# 20

# *Remote Login: TELNET and SSH*

The main task of the Internet and its TCP/IP protocol suite is to provide services for users. Although there are some specific client/server programs that we discuss in future chapters, it would be impossible to write a specific client-server program for each demand. The better solution is a general-purpose client-server program that lets a user access any application program on a remote computer; in other words, allow the user to log on to a remote computer. After logging on, a user can use the services available on the remote computer and transfer the results back to the local computer. In this chapter, we discuss two of these application programs: TELNET and SSH.

## OBJECTIVES

*The chapter has several objectives:*

- To introduce the TELNET protocol and show how it implements local and remote login using the concept of network virtual terminal.
- To discuss options and suboptions used in TELNET and how they are negotiated.
- To define out-of-band signaling and how it is implemented in TELNET.
- To define different modes of operations in TELNET and their efficiency.
- To introduce SSH as an alternative to TELNET.
- To show how different components of SSH are combined to provide a secure connection over an insecure TCP connection.
- To discuss port-forwarding in SSH and how it can be used to provide security for other applications.

## 20.1 TELNET

**TELNET** is an abbreviation for *TErminaL NETwork*. It is the standard TCP/IP protocol for virtual terminal service as proposed by ISO. TELNET enables the establishment of a connection to a remote system in such a way that the local terminal appears to be a terminal at the remote system.

**TELNET is a general-purpose client-server application program.**

### Concepts

TELNET is related to several concepts that we briefly describe here.

### Time-Sharing Environment

TELNET was designed at a time when most operating systems, such as UNIX, were operating in a **time-sharing** environment. In such an environment, a large computer supports multiple users. The interaction between a user and the computer occurs through a terminal, which is usually a combination of keyboard, monitor, and mouse. Even a microcomputer can simulate a terminal with a terminal emulator.

In a time-sharing environment, all of the processing must be done by the central computer. When a user types a character on the keyboard, the character is usually sent to the computer and echoed to the monitor. Time-sharing creates an environment in which each user has the illusion of a dedicated computer. The user can run a program, access the system resources, switch from one program to another, and so on.

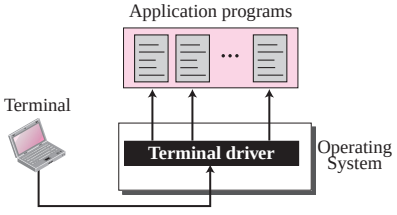
#### *Login*

In a time-sharing environment, users are part of the system with some right to access resources. Each authorized user has an identification and probably a password. The user identification defines the user as part of the system. To access the system, the user logs into the system with a user id or login name. The system also includes password checking to prevent an unauthorized user from accessing the resources.

**Local Login** When a user logs into a local time-sharing system, it is called **local login.** As a user types at a terminal or at a workstation running a terminal emulator, the keystrokes are accepted by the terminal driver. The terminal driver passes the characters to the operating system. The operating system, in turn, interprets the combination of characters and invokes the desired application program or utility (see Figure 20.1).

The mechanism, however, is not as simple as it seems because the operating system may assign special meanings to special characters. For example, in UNIX some

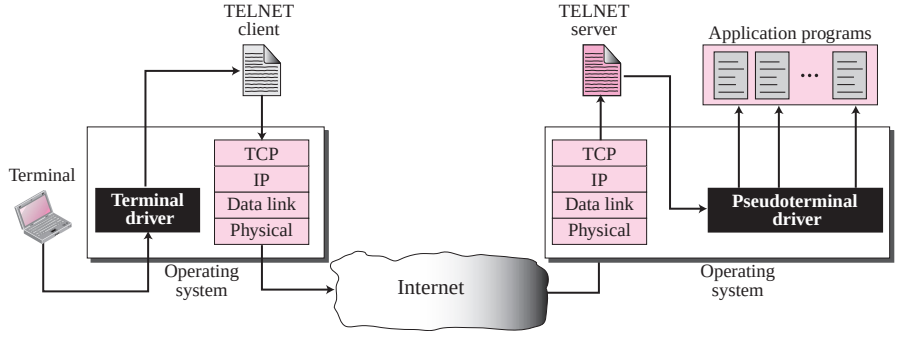
**Figure 20.1** *Local login*




combinations of characters have special meanings, such as the combination of the control character with the character z, which means suspend; the combination of the control character with the character c, which means abort; and so on. Whereas these special situations do not create any problem in local login because the terminal emulator and the terminal driver know the exact meaning of each character or combination of characters, they may create problems in remote login. Which process should interpret special characters? The client or the server? We will clarify this situation later in the chapter.

**Remote Login** When a user wants to access an application program or utility located on a remote machine, he or she performs **remote login.** Here the TELNET client and server programs come into use. The user sends the keystrokes to the terminal driver where the local operating system accepts the characters but does not interpret them. The characters are sent to the TELNET client, which transforms the characters to a universal character set called *Network Virtual Terminal* (NVT) *characters* and delivers them to the local TCP/IP stack (see Figure 20.2).

**Figure 20.2** *Remote login*




The commands or text, in NVT form, travel through the Internet and arrive at the TCP/IP stack at the remote machine. Here the characters are delivered to the operating system and passed to the TELNET server, which changes the characters to the corresponding characters understandable by the remote computer. However, the characters cannot be passed directly to the operating system because the remote operating system

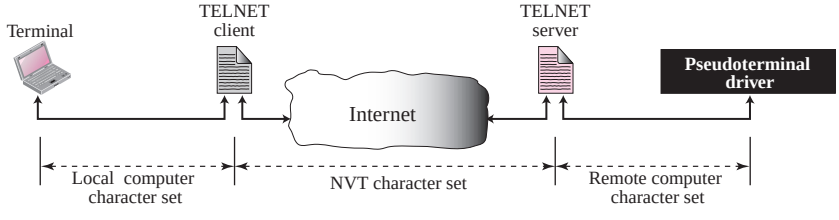
is not designed to receive characters from a TELNET server: It is designed to receive characters from a terminal driver. The solution is to add a piece of software called a *pseudoterminal driver*, which pretends that the characters are coming from a terminal. The operating system then passes the characters to the appropriate application program.

## Network Virtual Terminal (NVT)

The mechanism to access a remote computer is complex. This is because every computer and its operating system accepts a special combination of characters as tokens. For example, the end-of-file token in a computer running the DOS operating system is Ctrl+z, while the UNIX operating system recognizes Ctrl+d.

We are dealing with heterogeneous systems. If we want to access any remote computer in the world, we must first know what type of computer we will be connected to, and we must also install the specific terminal emulator used by that computer. TELNET solves this problem by defining a universal interface called the **Network Virtual Terminal (NVT)** character set. Via this interface, the client TELNET translates characters (data or commands) that come from the local terminal into NVT form and delivers them to the network. The server TELNET, on the other hand, translates data and commands from NVT form into the form acceptable by the remote computer. For an illustration of this concept, see Figure 20.3.

**Figure 20.3** *Concept of NVT*




### NVT Character Set

NVT uses two sets of characters, one for data and one for control. Both are 8-bit bytes (Figure 20.4).

**Figure 20.4** *Format of data and control characters*


a. Data Character b. Control Character

**Data Characters** For data, NVT normally uses what is called NVT ASCII. This is an 8-bit character set in which the seven lowest order bits are the same as US ASCII and the highest order bit is 0 (see Figure 20.4). Although it is possible to send an 8-bit

ASCII (with the highest order bit set to be 0 or 1), this must first be agreed upon between the client and the server using option negotiation.

**Control Characters** To send **control characters** between computers (from client to server or vice versa), NVT uses an 8-bit character set in which the highest order bit is set to 1 (see Figure 20.4). Table 20.1 lists some of the control characters and their meanings. Later we will categorize these control characters on the basis of their functionalities.

**Table 20.1** *Some NVT control characters*

| *Character* | *Decimal* | *Binary* | *Meaning* |
|---|---|---|---|
| EOF | 236 | 11101100 | End of file |
| EOR | 239 | 11101111 | End of record |
| SE | 240 | 11110000 | Suboption end |
| NOP | 241 | 11110001 | No operation |
| DM | 242 | 11110010 | Data mark |
| BRK | 243 | 11110011 | Break |
| IP | 244 | 11110100 | Interrupt process |
| AO | 245 | 11110101 | Abort output |
| AYT | 246 | 11110110 | Are you there? |
| EC | 247 | 11110111 | Erase character |
| EL | 248 | 11111000 | Erase line |
| GA | 249 | 11111001 | Go ahead |
| SB | 250 | 11111010 | Suboption begin |
| WILL | 251 | 11111011 | Agreement to enable option |
| WONT | 252 | 11111100 | Refusal to enable option |
| DO | 253 | 11111101 | Approval to option request |
| DONT | 254 | 11111110 | Denial of option request |
| IAC | 255 | 11111111 | Interpret (the next character) as control |

## Embedding

TELNET uses only one TCP connection. The server uses the well-known port 23 and the client uses an ephemeral port. The same connection is used for sending both data and control characters. TELNET accomplishes this by embedding the control characters in the data stream. However, to distinguish data from control characters, each sequence of control characters is preceded by a special control character called *interpret as control* (IAC). For example, imagine a user wants a server to display a file (*file1*) on a remote server. She types:

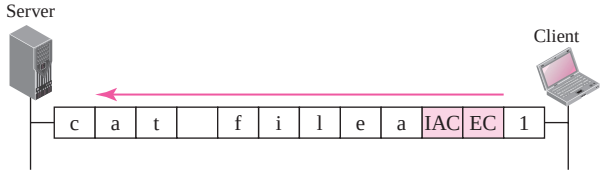
```
cat file1
```

in which *cat* is a Unix command that displays the content of the file on the screen. However, the name of the file has been mistyped (*filea* instead of *file1*). The user uses the backspace key to correct this situation.

```
cat filea<backspace>1
```

However, in the default implementation of TELNET, the user cannot edit locally; the editing is done at the remote server. The backspace character is translated into two remote characters (IAC EC), which is embedded in the data and sent to the remote server. What is sent to the server is shown in Figure 20.5.

**Figure 20.5** *An example of embedding*




## Options

TELNET lets the client and server negotiate options before or during the use of the service. Options are extra features available to a user with a more sophisticated terminal. Users with simpler terminals can use default features. Some control characters discussed previously are used to define options. Table 20.2 shows some common options.

**Table 20.2** *Options*

| *Code* | *Option* | *Meaning* |
|---|---|---|
| 0 | Binary | Interpret as 8-bit binary transmission |
| 1 | Echo | Echo the data received on one side to the other |
| 3 | Suppress go-ahead | Suppress go-ahead signals after data |
| 5 | Status | Request the status of TELNET |
| 6 | Timing mark | Define the timing marks |
| 24 | Terminal type | Set the terminal type |
| 32 | Terminal speed | Set the terminal speed |
| 34 | Line mode | Change to line mode |

The option descriptions are as follows:

- **Binary.** This option allows the receiver to interpret every 8-bit character received, except IAC, as binary data. When IAC is received, the next character or characters are interpreted as commands. However, if two consecutive IAC characters are received, the first is discarded and the second is interpreted as data.
- **Echo.** This option allows the server to echo data received from the client. This means that every character sent by the client to the sender will be echoed back to the screen of the client terminal. In this case, the user terminal usually does not echo characters when they are typed but waits until it receives them from the server.
- **Suppress go-ahead.** This option suppresses the go-ahead (GA) character (see section on Modes of Operation).
- **Status.** This option allows the user or the process running on the client machine to get the status of the options being enabled at the server site.

- **Timing mark.** This option allows one party to issue a timing mark that indicates all previously received data has been processed.
- **Terminal type.** This option allows the client to send its terminal type.
- **Terminal speed.** This option allows the client to send its terminal speed.
- **Line mode.** This option allows the client to switch to the line mode. We will discuss the line mode later.

### *Option Negotiation*

To use any of the options mentioned in the previous section first requires **option negotiation** between the client and the server. Four control characters are used for this purpose; these are shown in Table 20.3.

**Table 20.3** *NVT character set for option negotiation*

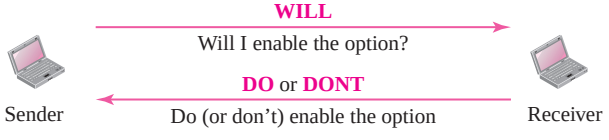
| *Character* | *Code* | *Meaning 1* | *Meaning 2* | *Meaning 3* |
|---|---|---|---|---|
| WILL | 251 | Offering to enable | Accepting to enable | |
| WONT | 252 | Rejecting to enable | Offering to disable | Accepting to disable |
| DO | 253 | Approving to enable | Requesting to enable | |
| DONT | 254 | Disapproving to enable | Approving to disable | Requesting to disable |

### *Enabling an Option*

Some options can only be enabled by the server, some only by the client, and some by both. An option is enabled either through an *offer* or a *request*.

**Offer to Enable** A party can offer to enable an option if it has the right to do so. The offering can be approved or disapproved by the other party. The offering party sends the *WILL* command, which means "Will I enable the option?" The other party sends either the *DO* command, which means "Please do," or the *DONT* command, which means "Please don't." See Figure 20.6.

**Figure 20.6** *Offer to enable an option*

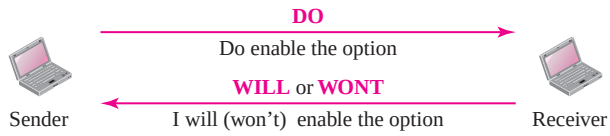



**Request to Enable** A party can request from the other party the enabling of an option. The request can be accepted or refused by the other party. The requesting party sends the *DO* command, which means "Please do enable the option." The other party sends either the *WILL* command, which means "I will," or the *WONT* command, which means "I won't." See Figure 20.7.

### *Disabling an Option*

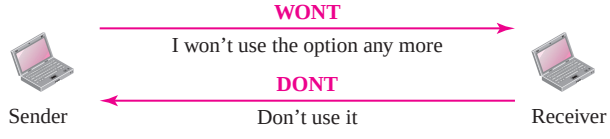
An option that has been enabled can be disabled by one of the parties. An option is disabled either through an *offer* or a *request*.

**Figure 20.7** *Request to enable an option*



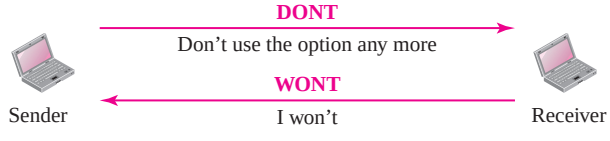

**Offer to Disable** A party can offer to disable an option. The other party must approve the offering; it cannot be disapproved. The offering party sends the *WONT* command, which means "I won't use this option any more." The answer must be the *DONT* command, which means "Don't use it anymore." Figure 20.8 shows an offer to disable an option.

**Figure 20.8** *Offer to disable an option*




**Request to Disable** A party can request from another party the disabling of an option. The other party must accept the request; it cannot be rejected. The requesting party sends the *DONT* command, which means "Please don't use this option anymore." The answer must be the *WONT* command, which means "I won't use it anymore." Figure 20.9 shows a request to disable an option.

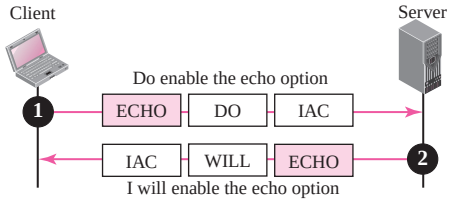
**Figure 20.9** *Request to disable an option*




### Example 20.1

Figure 20.10 shows an example of option negotiation. In this example, the client wants the server to echo each character sent to the server. In other words, when a character is typed at the user keyboard terminal, it goes to the server and is sent back to the screen of the user before being processed. The echo option is enabled by the server because it is the server that sends the characters back to the user terminal. Therefore, the client should *request* from the server the enabling of the option using DO. The request consists of three characters: IAC, DO, and ECHO. The server accepts the request and enables the option. It informs the client by sending the three-character approval: IAC, WILL, and ECHO.

**Figure 20.10** *Example 20.1: Echo option*




## Symmetry

One interesting feature of TELNET is its symmetric option negotiation in which the client and server are given equal opportunity. This means that, at the beginning of connection, it is assumed that both sides are using a default TELNET implementation with no options enabled. If one party wants an option enabled, it can offer or request. The other party has the right to approve the offer or reject the request if the party is not capable of using the option or does not want to use the option. This allows for the expansion of TELNET. A client or server can install a more sophisticated version of TELNET with more options. When it is connected to a party, it can offer or request these new options. If the other party also supports these options, the options can be enabled; otherwise, they are rejected.

## Suboption Negotiation

Some options require additional information. For example, to define the type or speed of a terminal, the negotiation includes a string or a number to define the type or speed. In either case, the two suboption characters indicated in Table 20.4 are needed for **suboption negotiation.**

**Table 20.4** *NVT character set for suboption negotiation*

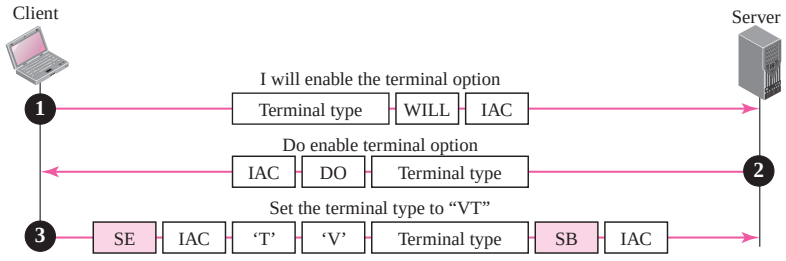
| *Character* | *Decimal* | *Binary* | *Meaning* |
|---|---|---|---|
| SE | 240 | 11110000 | Suboption end |
| SB | 250 | 11111010 | Suboption begin |

For example, the type of the terminal is set by the client, as is shown in Figure 20.11.

## Controlling the Server

Some control characters can be used to control the remote server. When an application program is running on the local computer, special characters are used to interrupt (abort) the program (for example, Ctrl+c), or erase the last character typed (for example, delete key or backspace key), and so on. However, when a program is running on a remote computer, these control characters are sent to the remote machine. The user still types the same sequences, but they are changed to special characters and sent to the

**Figure 20.11** *Example of sub-option negotiation*




server. Table 20.5 shows some of the characters that can be sent to the server to control the application program that is running there.

**Table 20.5** *Characters used to control a program running on remote server*

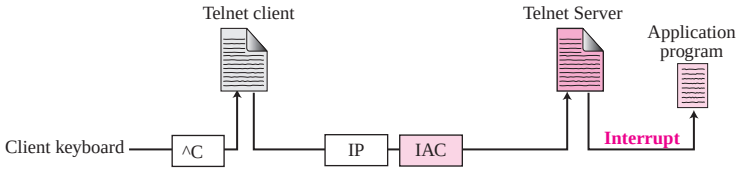
| *Character* | *Decimal* | *Binary* | *Meaning* |
|---|---|---|---|
| IP | 244 | 11110100 | Interrupt process |
| AO | 245 | 11110101 | Abort output |
| AYT | 246 | 11110110 | Are you there? |
| EC | 247 | 11110111 | Erase the last character |
| EL | 248 | 11111000 | Erase line |

Let's look at these characters in more detail:

- **IP (interrupt process).** When a program is being run locally, the user can interrupt (abort) the program if, for example, the program has gone into an infinite loop. The user can type the Ctrl+c combination, the operating system calls a function, and the function aborts the program. However, if the program is running on a remote machine, the appropriate function should be called by the operating system of the remote machine. TELNET defines the IP (interrupt process) control character that is read and interpreted as the appropriate command for invoking the interrupting function in the remote machine.
- **AO (abort output).** This is the same as IP, but it allows the process to continue without creating output. This is useful if the process has another effect in addition to creating output. The user wants this effect but not the output. For example, most commands in UNIX generate output and have an exit status. The user may want the exit status for future use but is not interested in the output data.
- **AYT (are you there?).** This control character is used to determine if the remote machine is still up and running, especially after a long silence from the server. When this character is received, the server usually sends an audible or visual signal to confirm that it is running.
- **EC (erase character).** When a user sends data from the keyboard to the local machine, the delete or backspace character can erase the last character typed. To do the same in a remote machine, TELNET defines the EC control character.
- **EL (erase line).** This is used to erase the current line in the remote host.

For example, Figure 20.12 shows how to interrupt a runaway application program at the server site. The user types Ctrl+c, but the TELNET client sends the combination of IAC and IP to the server.

**Figure 20.12** *Example of interrupting an application program*




## Out-of-Band Signaling

To make control characters effective in special situations, TELNET uses **out-of-band signaling.** In out-of-band signaling, the control characters are preceded by IAC and are sent to the remote process.

Imagine a situation in which an application program running at the server site has gone into an infinite loop and does not accept any more input data. The user wants to interrupt the application program, but the program does not read data from the buffer. The TCP at the server site has found that the buffer is full and has sent a segment specifying that the client window size should be zero. In other words, the TCP at the server site is announcing that no more regular traffic is accepted. To remedy such a situation, an urgent TCP segment should be sent from the client to the server. The urgent segment overrides the regular flow-control mechanism. Although TCP is not accepting normal segments, it must accept an urgent segment.

When a TELNET process (client or server) wants to send an out-of-band sequence of characters to the other process (client or server), it embeds the sequence in the data stream and inserts a special character called a DM (data mark). However, to inform the other party, it creates a TCP segment with the urgent bit set and the urgent pointer pointing to the DM character. When the receiving process receives the data, it reads the data and discards any data preceding the control characters (IAC and IP, for example). When it reaches the DM character, the remaining data are handled normally. In other words, the DM character is used as a *synchronization* character that switches the receiving process from the urgent mode to the normal mode and *resynchronizes* the two ends (see Figure 20.13).

In this way, the control character (IP) is delivered out of band to the operating system, which uses the appropriate function to interrupt the running application program.

## Escape Character

A character typed by the user is normally sent to the server. However, sometimes the user wants characters interpreted by the client instead of the server. In this case, the user can use an *escape* character, normally Ctrl+] (shown as ^]). Figure 20.14 compares the interruption of an application program at the remote site with the interruption of the

**Figure 20.13** *Out-of-band signaling*

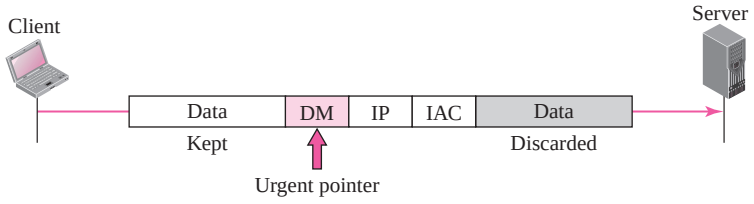
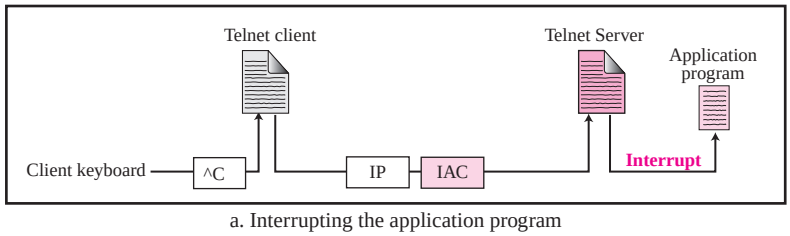
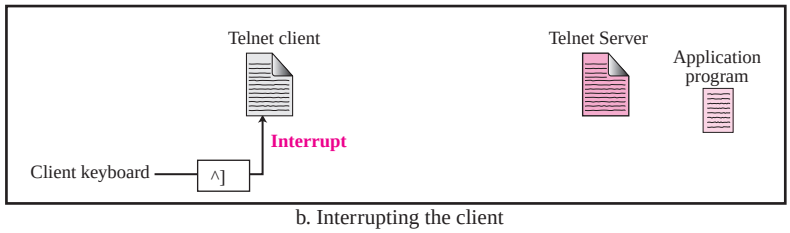



**Figure 20.14** *Two different interruptions*




a. Interrupting the application program




b. Interrupting the client

client process at the local site using the escape character. The TELNET prompt is displayed after this escape character.

## Modes of Operation

Most TELNET implementations operate in one of three modes: default mode, character mode, or line mode.

### *Default Mode*

The **default mode** is used if no other modes are invoked through option negotiation. In this mode, the echoing is done by the client. The user types a character and the client echoes the character on the screen (or printer) but does not send it until a whole line is completed. After sending the whole line to the server, the client waits for the GA (go ahead) command from the server before accepting a new line from the user. The operation is half-duplex. Half-duplex operation is not efficient when the TCP connection itself is full-duplex, and so this mode is becoming obsolete.

### *Character Mode*

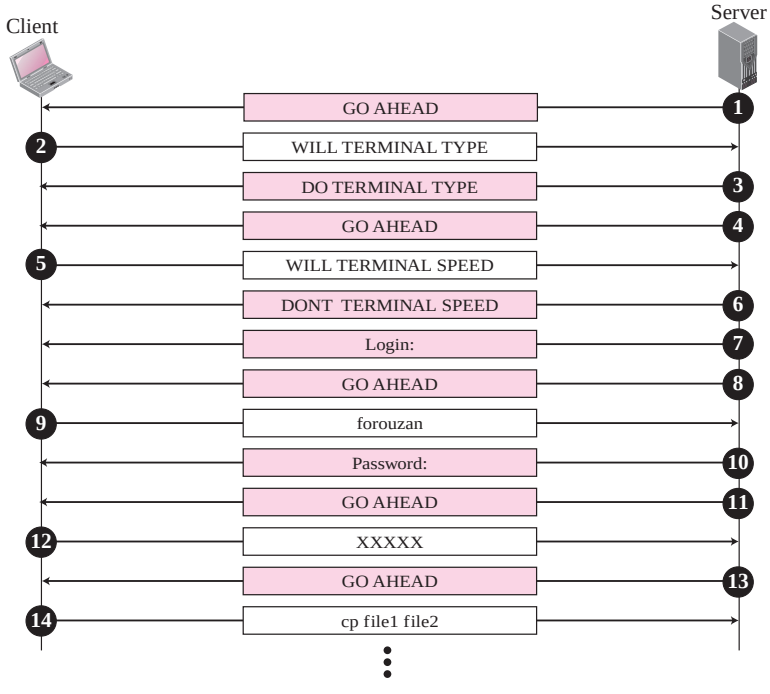
In the **character mode,** each character typed is sent by the client to the server. The server normally echoes the character back to be displayed on the client screen. In this mode the echoing of the character can be delayed if the transmission time is long (such as in a satellite connection). It also creates overhead (traffic) for the network because three TCP segments must be sent for each character of data:

1. The user enters a character that is sent to the server.
2. The server acknowledges the received character and echoes the character back (in one segment).
3. The client acknowledges the receipt of the echoed character.

### Example 20.2

In this example, we use the default mode to show the concept and its deficiencies even though it is almost obsolete today. The client and the server negotiate the terminal type and terminal speed and then the server checks the login and password of the user (see Figure 20.15).

**Figure 20.15** *Example 20.2*




### *Line Mode*

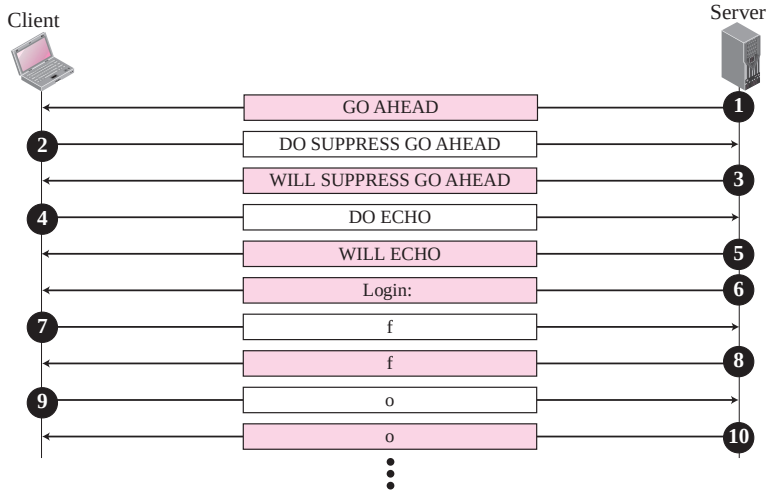
A new mode has been proposed to compensate for the deficiencies of the default mode and the character mode. In this mode, called the **line mode,** line editing (echoing, character erasing, line erasing, and so on) is done by the client. The client then sends

the whole line to the server. Although the line mode looks like the default mode, it is not. The default mode operates in the half-duplex mode; the line mode is full-duplex with the client sending one line after another, without the need for an intervening GA (go ahead) character from the server.

### Example 20.3

In this example, we show how the client switches to the character mode. This requires that the client request the server to enable the SUPPRESS GO AHEAD and ECHO options (see Figure 20.16).

**Figure 20.16** *Example 20.3*




## User Interface

The normal user does not use TELNET commands as defined above. Usually, the operating system (UNIX, for example) defines an interface with user-friendly commands. An example of such a set of commands can be found in Table 20.6. Note that the interface is responsible for translating the user-friendly commands to the previously defined commands in the protocol.

**Table 20.6** *Examples of interface commands*

| *Command* | *Meaning* | *Command* | *Meaning* |
|---|---|---|---|
| open | Connect to a remote computer | set | Set the operating parameters |
| close | Close the connection | status | Display the status information |
| display | Show the operating parameters | send | Send special characters |
| mode | Change to line or character mode | quit | Exit TELNET |

### Security Issue

TELNET suffers from security problems. Although TELNET requires a login name and password (when exchanging text), often this is not enough. A microcomputer connected to a broadcast LAN can easily eavesdrop using snooper software and capture a login name and the corresponding password (even if it is encrypted). In Chapter 29, we will learn more about authentication and security.

## 20.2 SECURE SHELL (SSH)

Another popular remote login application program is **Secure Shell (SSH)**. SSH, like TELNET, uses TCP as the underlying transport protocol, but SSH is more secure and provides more services than TELNET.

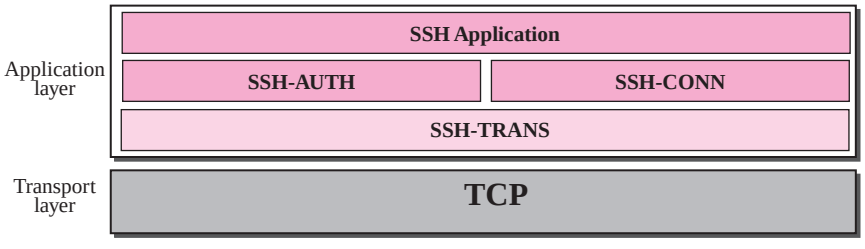
### Versions

There are two versions of SSH: SSH-1 and SSH-2, which are totally incompatible. The first version, SSH-1 is now deprecated because of security flaws in it. In this section, we discuss only SSH-2.

### Components

SSH is a proposed application-layer protocol with four components, as shown in Figure 20.17.

**Figure 20.17** *Components of SSH*




#### *SSH Transport-Layer Protocol (SSH-TRANS)*

Since TCP is not a secured transport layer protocol, SSH first uses a protocol that creates a secured channel on the top of TCP. This new layer is an independent protocol referred to as SSH-TRANS. When the software implementing this protocol is called, the client and server first use the TCP protocol to establish an insecure proconnection. Then they exchange several security parameters to establish a secure channel on the top of the TCP. We discuss network security in Chapter 29, but we briefly list the services provided by this protocol:

1. Privacy or confidentiality of the message exchanged.
2. Data integrity, which means that it is guaranteed that the messages exchanged between the client and server are not changed by an intruder.

3. Server authentication, which means that the client is now sure that the server is the one that it claims to be.
4. Compression of the messages that improve the efficiency of the system and makes attack more difficult.

### *SSH Authentication Protocol (SSH-AUTH)*

After a secure channel is established between the client and the server and the server is authenticated for the client, SSH can call another software that can authenticate the client for the server.

### *SSH Connection Protocol (SSH-CONN)*

After the secured channel is established and both server and client are authenticated for each other, SHH can call a piece of software that implements the third protocol, SSH-CONN. One of the services provided by the SSH-CONN protocol is to do multiplexing. SSH-CONN takes the secure channel established by the two previous protocols and lets the client create multiple logical channels over it.

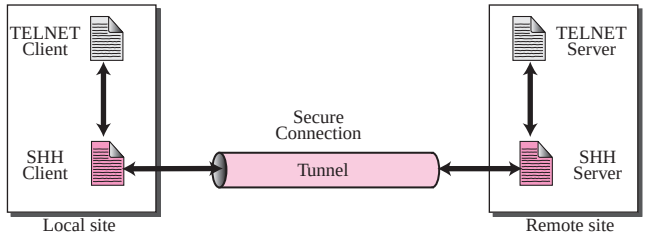
### *SSH Applications*

After the connection phase is completed, SSH allows several application programs to use the connection. Each application can create a logical channel as described above and then benefit from the secured connection. In other words, remote login is one of the services that can use the SSH-CONN protocols; other applications, such as a file transfer application can use one of the logical channels for this purpose. In the next chapter, we show how SSH can be used for secure file transfer.

## Port Forwarding

One of the interesting services provided by the SSH protocol is to provide **port forwarding**. We can use the secured channels available in SSH to access an application program that does not provide security services. Application such as TELNET (see Chapter 20) and SMTP (see Chapter 23) can use the services of SSH using port forwarding mechanism. SSH port forwarding mechanism creates a tunnel through which the messages belonging to other protocol can travel. For this reason, this mechanism is sometimes referred to as SSH **tunneling**. Figure 20.18 shows the concept of port forwarding.

**Figure 20.18** *Port Forwarding*

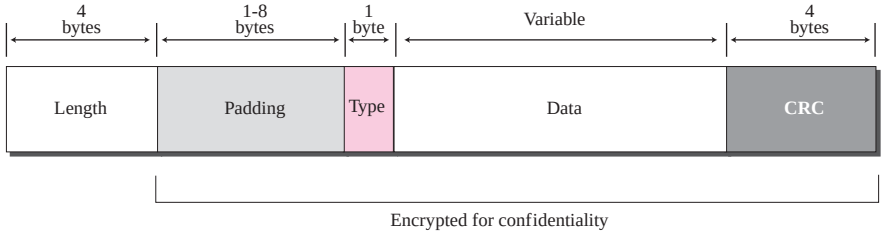


We can change a direct, but insecure, connection between the TELNET client and the TELNET server by port forwarding. The TELNET client can use the SHH client on the local site to make a secure connection with the SSH server on the remote site. Any request from the TELNET client to the TELNET server is carried through the tunnel provided by the SSH client and server. Any response from the TELNET server to the TELNET client is also carried through the tunnel provided by the SSH client and server. We talk more about tunneling in Chapter 30.

### Format of the SSH Packets

Figure 20.19 shows the format of packets used by the SSH protocols.

**Figure 20.19** *SSH Packet Format*




The following is the brief description of each field:

- **Length.** This 4-byte field defines the length of the packet including the type, the data, and the CRC field, but not the padding and the length field.
- **Padding.** One to eight bytes of padding is added to the packet to make the attack on the security provision more difficult.
- **Type.** This one-byte field defines the type of the packet used by SSH protocols.
- **Data.** This field is of variable length. The length of the data can be found by deducting the five bytes from the value of the length field.
- **CRC.** The cyclic redundancy check filed is used for error detection (see Appendix D).

## 20.3 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books and RFCs give an easy but thorough coverage of TELNET and SSH including [Com 06], [Mir 07], [Bar et al. 05].

### RFCs

Several RFCs show different updates on TELNET including RFC 854, FRC 855, RFC 856, RFC 1041, RFC 1091, RFC 1372, and RFC 1572. More information about SSH can be found in RFC 4250, RFC 4251, RFC 4252, RFC 4253, RFC 4254, and RFC 4344.

## 20.4 KEY TERMS

character mode
control character
default mode
line mode
local login
Network Virtual Terminal (NVT)
option negotiation
out-of-band signaling
port forwarding
remote login
Secure Shell (SHH)
suboption negotiation
terminal network (TELNET)
time-sharing
tunneling

## 20.5 SUMMARY

- TELNET is a client-server application that allows a user to log on to a remote machine, giving the user access to the remote system. When a user accesses a remote system via the TELNET process, this is comparable to a time-sharing environment. A terminal driver correctly interprets the keystrokes on the local terminal or terminal emulator. This may not occur between a terminal and a remote terminal driver.
- TELNET uses the Network Virtual Terminal (NVT) system to encode characters on the local system. On the server machine, NVT decodes the characters to a form acceptable to the remote machine. NVT uses a set of characters for data and a set of characters for control.
- Options are features that enhance the TELNET process. TELNET allows negotiation to set transfer conditions between the client and server before and during the use of the service. Some options can only be enabled by the server, some only by the client, and some by both. An option is enabled or disabled through an offer or a request. An option that needs additional information requires the use of suboption characters.
- A TELNET implementation operates in the default, character, or line mode. In the default mode, the client sends one line at a time to the server and waits for the go ahead (GA) character before a new line from the user can be accepted. In the character mode, the client sends one character at a time to the server. In the line mode, the client sends one line at a time to the server, one after the other, without the need for an intervening GA character.
- Another popular remote login application program is Secure Shell (SSH), which is more secure and provides more services than TELNET. There are two versions of SSH; we discussed only SSH-2.

- SSH is made of four components: SSH application, SSH-CONN, SSH-AUTH, and SSH-TRANS. The combination of the above four components provide a secure remote login that can be used instead of TELNET.
- One of the interesting services provided by the SSH protocol is to provide port forwarding. We can use the secured channels available in SSH to access an application program that does not provide security services.

## 20.6 PRACTICE SET

### Exercises

1. Show the sequence of bits sent from a client TELNET for the binary transmission of 11110011 00111100 11111111.
2. If TELNET is using the character mode, how many characters are sent back and forth between the client and server to copy a file named file1 to another file named file2 in UNIX (*cp file1 file2*)?
3. What is the minimum number of bits sent at the TCP level to accomplish the task in Exercise 1?
4. What is the minimum number of bits sent at the data link layer level (using Ethernet) to accomplish the task in Exercise 1?
5. What is the ratio of the useful bits to the total bits in Exercise 4?
6. Show the sequence of characters exchanged between the TELNET client and the server to switch from the default mode to the character mode.
7. Show the sequence of characters exchanged between the TELNET client and the server to switch from the character mode to the default mode.
8. Show the sequence of characters exchanged between the TELNET client and the server to switch from the default mode to line mode.
9. Show the sequence of characters exchanged between the TELNET client and the server to switch from the character mode to the line mode.
10. Show the sequence of characters exchanged between the TELNET client and the server to switch from the line mode to the character mode.
11. Show the sequence of characters exchanged between the TELNET client and the server to switch from the line mode to the default mode.
12. Interpret the following sequence of characters (in hexadecimal) received by a TELNET client or server:
    a. FF FB 01
    b. FF FE 01
    c. FF F4
    d. FF F9
13. If you know a site that allows you to use SSH server, create a connection with the site by using the client SSH on your computer.

## Research Activities

**14.** Find the extended options proposed for TELNET.

**15.** SSH provide two types of port forwarding: local and remote. Do some research and find the difference between the two.

**16.** Another login protocol is called Rlogin. Find some information about Rlogin and compare it with TELNET and SSH.

**17.** Another login protocol is called Remote Desktop Protocol (RDP). Find some information about RDP and compare it with TELNET and SSH.

**18.** Virtual Network Computing (VNC) is a program that provides remote desktop capability. Find some information about VNC and compare it with TELNET and SSH.

CHAPTER

# 21

# File Transfer: FTP and TFTP

Transferring files from one computer to another is one of the most common tasks expected from a networking or internetworking environment. As a matter of fact, the greatest volume of data exchange in the Internet today is due to file transfer. In this chapter, we discuss two protocols involved in transferring files: File Transfer Protocol (FTP) and Trivial File Transfer Protocol (TFTP).

## OBJECTIVES

*The chapter has several objectives:*

- To discuss FTP and two connections used in this protocol: control connection and data connection.
- To discuss six classes of commands sent by the client to establish communication with the server.
- To explain three types of file transfer transferred by FTP.
- To show some user-friendly commands used by some FTP interfaces.
- To discuss anonymous FTP and its application.
- To discuss how file transfer can be done using a secure channel.
- To discuss TFTP as a simple file transfer protocol without the complexities and sophistication of FTP.
- To discuss five types of TFTP messages and their applications.
- To discuss the sorcerer's apprentice bug related to TFTP's flow- and error-control mechanisms.
- To show how TFTP can be used in conjunction with DHCP to initialize devices by downloading configuration files.

## 21.1 FTP

**File Transfer Protocol (FTP)** is the standard mechanism provided by TCP/IP for copying a file from one host to another. Although transferring files from one system to another seems simple and straightforward, some problems must be dealt with first. For example, two systems may use different file name conventions. Two systems may have different ways to represent text and data. Two systems may have different directory structures. All of these problems have been solved by FTP in a very simple and elegant approach.

FTP differs from other client-server applications in that it establishes two connections between the hosts. One connection is used for data transfer, the other for control information (commands and responses). Separation of commands and data transfer makes FTP more efficient. The control connection uses very simple rules of communication. We need to transfer only a line of command or a line of response at a time. The data connection, on the other hand, needs more complex rules due to the variety of data types transferred.

FTP uses two well-known TCP ports: Port 21 is used for the control connection, and port 20 is used for the data connection.

**FTP uses the services of TCP. It needs two TCP connections. The well-known port 21 is used for the control connection and the well-known port 20 for the data connection.**

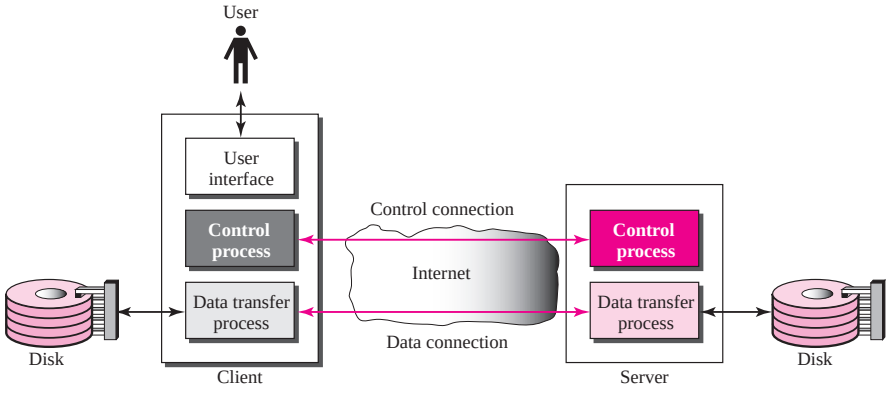
Figure 21.1 shows the basic model of FTP. The client has three components: user interface, client control process, and the client data transfer process. The server has two components: the server control process and the server data transfer process. The control connection is made between the control processes. The data connection is made between the data transfer processes.

The **control connection** remains connected during the entire interactive FTP session. The data connection is opened and then closed for each file transferred. It opens each time commands that involve transferring files are used, and it closes when the file is transferred. In other words, when a user starts an FTP session, the control connection opens. While the control connection is open, the data connection can be opened and closed multiple times if several files are transferred.

### Connections

The two FTP connections, control and data, use different strategies and different port numbers.

**Figure 21.1** *FTP*



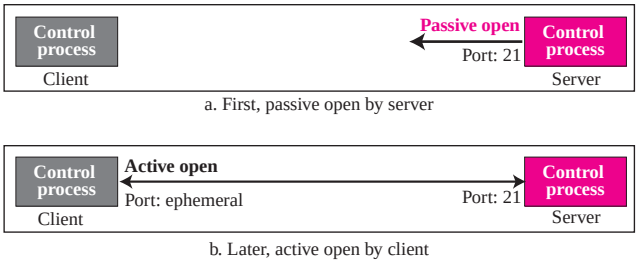

### *Control Connection*

The control connection is created in the same way as other application programs described so far. There are two steps:

1. The server issues a passive open on the well-known port 21 and waits for a client.
2. The client uses an ephemeral port and issues an active open.

The connection remains open during the entire process. The service type, used by the IP protocol, is *minimize delay* because this is an interactive connection between a user (human) and a server. The user types commands and expects to receive responses without significant delay. Figure 21.2 shows the initial connection between the server and the client.

**Figure 21.2** *Opening the control connection*




### *Data Connection*

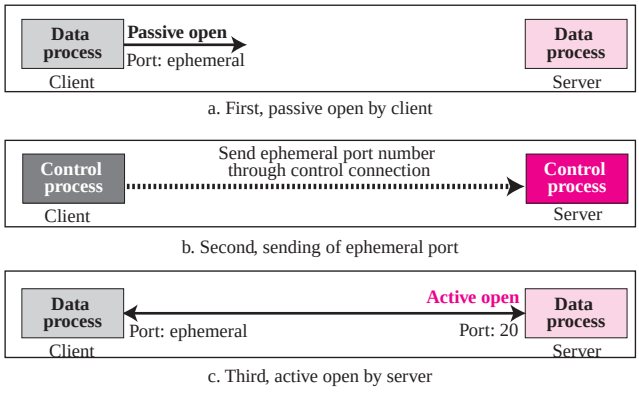
The **data connection** uses the well-known port 20 at the server site. However, the creation of a data connection is different from what we have seen so far. The following shows how FTP creates a data connection:

1. The client, not the server, issues a passive open using an ephemeral port. This must be done by the client because it is the client that issues the commands for transferring files.

2. The client sends this port number to the server using the PORT command (we will discuss this command shortly).
3. The server receives the port number and issues an active open using the well-known port 20 and the received ephemeral port number.

The steps for creating the initial data connection are shown in Figure 21.3. Later we will see that these steps are changed if the PASV command is used.

**Figure 21.3** *Creating the data connection*




## Communication

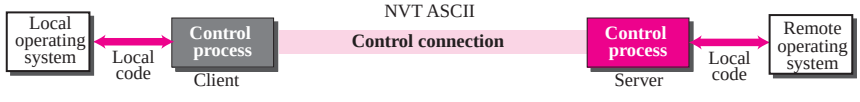
The FTP client and server, which run on different computers, must communicate with each other. These two computers may use different operating systems, different character sets, different file structures, and different file formats. FTP must make this heterogeneity compatible.

FTP has two different approaches, one for the control connection and one for the data connection. We will study each approach separately.

### *Communication over Control Connection*

FTP uses the same approach as TELNET or SMTP to communicate across the control connection. It uses the NVT ASCII character set (see Figure 21.4). Communication is achieved through commands and responses. This simple method is adequate for the control connection because we send one command (or response) at a time. Each command or response is only one short line so we need not worry about file format or file

**Figure 21.4** *Using the control connection*

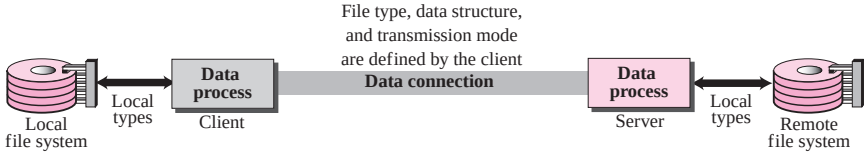


structure. Each line is terminated with a two-character (carriage return and line feed) end-of-line token.

### *Communication over Data Connection*

The purpose and implementation of the data connection are different from that of the control connection. We want to transfer files through the data connection. The client must define the type of file to be transferred, the structure of the data, and the transmission mode. Before sending the file through the data connection, we prepare for transmission through the control connection. The heterogeneity problem is resolved by defining three attributes of communication: file type, data structure, and transmission mode (see Figure 21.5).

**Figure 21.5** *Using the data connection*




**File Type** FTP can transfer one of the following file types across the data connection:

- **ASCII file.** This is the default format for transferring text files. Each character is encoded using NVT ASCII. The sender transforms the file from its own representation into NVT ASCII characters and the receiver transforms the NVT ASCII characters to its own representation.
- **EBCDIC file.** If one or both ends of the connection use EBCDIC encoding, the file can be transferred using EBCDIC encoding.
- **Image file.** This is the default format for transferring binary files. The file is sent as continuous streams of bits without any interpretation or encoding. This is mostly used to transfer binary files such as compiled programs.

If the file is encoded in ASCII or EBCDIC, another attribute must be added to define the printability of the file.

a. **Nonprint.** This is the default format for transferring a text file. The file contains no vertical specifications for printing. This means that the file cannot be printed without further processing because there are no characters to be interpreted for vertical movement of the print head. This format is used for files that will be stored and processed later.

b. **TELNET.** In this format the file contains NVT ASCII vertical characters such as CR (carriage return), LF (line feed), NL (new line), and VT (vertical tab). The file is printable after transfer.

**Data Structure** FTP can transfer a file across the data connection using one of the following interpretations about the structure of the data:

- **File structure (default).** The file has no structure. It is a continuous stream of bytes.

- **Record structure.** The file is divided into records. This can be used only with text files.
- **Page structure.** The file is divided into pages, with each page having a page number and a page header. The pages can be stored and accessed randomly or sequentially.

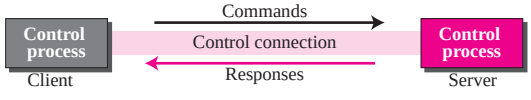
**Transmission Mode** FTP can transfer a file across the data connection using one of the following three transmission modes:

- **Stream mode.** This is the default mode. Data are delivered from FTP to TCP as a continuous stream of bytes. TCP is responsible for chopping data into segments of appropriate size. If the data is simply a stream of bytes (file structure), no end-of-file is needed. End-of-file in this case is the closing of the data connection by the sender. If the data are divided into records (record structure), each record will have a 1-byte end-of-record (EOR) character and the end of the file will have a 1-byte end-of-file (EOF) character.
- **Block mode.** Data can be delivered from FTP to TCP in blocks. In this case, each block is preceded by a 3-byte header. The first byte is called the *block descriptor;* the next two bytes define the size of the block in bytes.
- **Compressed mode.** If the file is big, the data can be compressed. The compression method normally used is run-length encoding. In this method, consecutive appearances of a data unit are replaced by one occurrence and the number of repetitions. In a text file, this is usually spaces (blanks). In a binary file, null characters are usually compressed.

## Command Processing

FTP uses the control connection to establish a communication between the client control process and the server control process. During this communication, the commands are sent from the client to the server and the responses are sent from the server to the client (see Figure 21.6).

**Figure 21.6** *Command processing*




### *Commands*

Commands, which are sent from the FTP client control process, are in the form of ASCII uppercase, which may or may not be followed by an argument. We can roughly divide the commands into six groups: access commands, file management commands, data formatting commands, port defining commands, file transferring commands, and miscellaneous commands.

- **Access commands.** These commands let the user access the remote system. Table 21.1 lists common commands in this group.

**Table 21.1** *Access commands*

| *Command* | *Argument(s)* | *Description* |
|---|---|---|
| **USER** | User id | User information |
| **PASS** | User password | Password |
| **ACCT** | Account to be charged | Account information |
| **REIN** | | Reinitialize |
| **QUIT** | | Log out of the system |
| **ABOR** | | Abort the previous command |

- **File management commands.** These commands let the user access the file system on the remote computer. They allow the user to navigate through the directory structure, create new directories, delete files, and so on. Table 21.2 gives common commands in this group.

**Table 21.2** *File management commands*

| *Command* | *Argument(s)* | *Description* |
|---|---|---|
| **CWD** | Directory name | Change to another directory |
| **CDUP** | | Change to parent directory |
| **DELE** | File name | Delete a file |
| **LIST** | Directory name | List subdirectories or files |
| **NLIST** | Directory name | List subdirectories or files without attributes |
| **MKD** | Directory name | Create a new directory |
| **PWD** | | Display name of current directory |
| **RMD** | Directory name | Delete a directory |
| **RNFR** | File name (old) | Identify a file to be renamed |
| **RNTO** | File name (new) | Rename the file |
| **SMNT** | File system name | Mount a file system |

- **Data formatting commands.** These commands let the user define the data structure, file type, and transmission mode. The defined format is then used by the file transfer commands. Table 21.3 shows common commands in this group.

**Table 21.3** *Data formatting commands*

| *Command* | *Argument(s)* | *Description* |
|---|---|---|
| **TYPE** | **A** (ASCII), **E** (EBCDIC), **I** (Image), **N** (Nonprint), or **T** (TELNET) | Define file type |
| **STRU** | **F** (File), **R** (Record), or **P** (Page) | Define organization of data |
| **MODE** | **S** (Stream), **B** (Block), or **C** (Compressed) | Define transmission mode |

- **Port defining commands.** These commands define the port number for the data connection on the client site. There are two methods to do this. In the first method, using the PORT command, the client can choose an ephemeral port number and send it to the server using a passive open. The server uses that port number and

creates an active open. In the second method, using the PASV command, the client just asks the server to first choose a port number. The server does a passive open on that port and sends the port number in the response (see response numbered 227 in Table 21.7). The client issues an active open using that port number. Table 21.4 shows the port defining commands.

**Table 21.4** *Port defining commands*

| *Command* | *Argument(s)* | *Description* |
|---|---|---|
| **PORT** | 6-digit identifier | Client chooses a port |
| **PASV** | | Server chooses a port |

- **File transfer commands.** These commands actually let the user transfer files. Table 21.5 lists common commands in this group.

**Table 21.5** *File transfer commands*

| *Command* | *Argument(s)* | *Description* |
|---|---|---|
| **RETR** | File name(s) | Retrieve files; file(s) are transferred from server to client |
| **STOR** | File name(s) | Store files; file(s) are transferred from client to server |
| **APPE** | File name(s) | Similar to STOR, but if file exists, data must be appended to it |
| **STOU** | File name(s) | Same as STOR, but file name will be unique in the directory |
| **ALLO** | File name(s) | Allocate storage space for files at the server |
| **REST** | File name(s) | Position file marker at a specified data point |
| **STAT** | File name(s) | Return status of files |

- **Miscellaneous commands.** These commands deliver information to the FTP user at the client site. Table 21.6 shows common commands in this group.

**Table 21.6** *Miscellaneous commands*

| *Command* | *Argument(s)* | *Description* |
|---|---|---|
| **HELP** | | Ask information about the server |
| **NOOP** | | Check if server is alive |
| **SITE** | Commands | Specify the site-specific commands |
| **SYST** | | Ask about operating system used by the server |

### *Responses*

Every FTP command generates at least one response. A response has two parts: a three-digit number followed by text. The numeric part defines the code; the text part defines needed parameters or extra explanations. We represent the three digits as *xyz*. The meaning of each digit is described below.

**First Digit** The first digit defines the status of the command. One of five digits can be used in this position:

- **1yz (positive preliminary reply).** The action has started. The server will send another reply before accepting another command.
- **2yz (positive completion reply).** The action has been completed. The server will accept another command.

- **3yz (positive intermediate reply).** The command has been accepted, but further information is needed.
- **4yz (transient negative completion reply).** The action did not take place, but the error is temporary. The same command can be sent later.
- **5yz (permanent negative completion reply).** The command was not accepted and should not be retried again.

**Second Digit** The second digit also defines the status of the command. One of six digits can be used in this position:

- **x0z (syntax).**
- **x1z (information).**
- **x2z (connections).**
- **x3z (authentication and accounting).**
- **x4z (unspecified).**
- **x5z (file system).**

**Third Digit** The third digit provides additional information. Table 21.7 shows a brief list of possible responses (using all three digits).

**Table 21.7** *Responses*

| *Code* | *Description* |
|---|---|
| **Positive Preliminary Reply** | |
| **120** | Service will be ready shortly |
| **125** | Data connection open; data transfer will start shortly |
| **150** | File status is OK; data connection will be open shortly |
| **Positive Completion Reply** | |
| **200** | Command OK |
| **211** | System status or help reply |
| **212** | Directory status |
| **213** | File status |
| **214** | Help message |
| **215** | Naming the system type (operating system) |
| **220** | Service ready |
| **221** | Service closing |
| **225** | Data connection open |
| **226** | Closing data connection |
| **227** | Entering passive mode; server sends its IP address and port number |
| **230** | User login OK |
| **250** | Request file action OK |
| **Positive Intermediate Reply** | |
| **331** | User name OK; password is needed |
| **332** | Need account for logging |
| **350** | The file action is pending; more information needed |

**Table 21.7** *Responses (continued)*

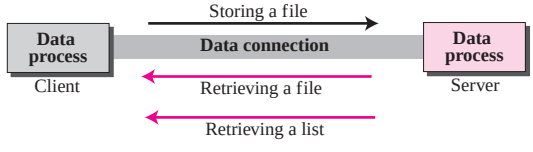
| *Code* | *Description* |
|---|---|
| **Transient Negative Completion Reply** | |
| **425** | Cannot open data connection |
| **426** | Connection closed; transfer aborted |
| **450** | File action not taken; file not available |
| **451** | Action aborted; local error |
| **452** | Action aborted; insufficient storage |
| **Permanent Negative Completion Reply** | |
| **500** | Syntax error; unrecognized command |
| **501** | Syntax error in parameters or arguments |
| **502** | Command not implemented |
| **503** | Bad sequence of commands |
| **504** | Command parameter not implemented |
| **530** | User not logged in |
| **532** | Need account for storing file |
| **550** | Action is not done; file unavailable |
| **552** | Requested action aborted; exceeded storage allocation |
| **553** | Requested action not taken; file name not allowed |

## File Transfer

File transfer occurs over the data connection under the control of the commands sent over the control connection. However, we should remember that file transfer in FTP means one of three things (see Figure 21.7).

- A file is to be copied from the server to the client (download). This is called *retrieving a file*. It is done under the supervision of the RETR command.
- A file is to be copied from the client to the server (upload). This is called *storing a file*. It is done under the supervision of the STOR command.
- A list of directory or file names is to be sent from the server to the client. This is done under the supervision of the LIST command. Note that FTP treats a list of directory or file names as a file. It is sent over the data connection.

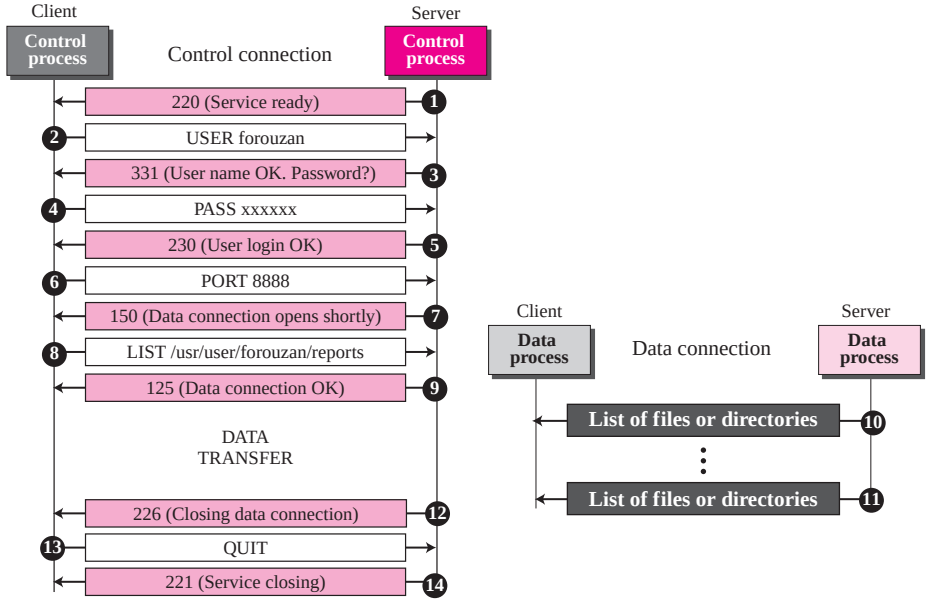
**Figure 21.7** *File transfer*




### Example 21.1

Figure 21.8 shows an example of using FTP for retrieving a list of items in a directory.

**Figure 21.8** *Example 21.1*




1. After the control connection to port 21 is created, the FTP server sends the 220 (service ready) response on the control connection.
2. The client sends the USER command.
3. The server responds with 331 (user name is OK, password is required).
4. The client sends the PASS command.
5. The server responds with 230 (user login is OK).
6. The client issues a passive open on an ephemeral port for the data connection and sends the PORT command (over the control connection) to give this port number to the server.
7. The server does not open the connection at this time, but it prepares itself for issuing an active open on the data connection between port 20 (server side) and the ephemeral port received from the client. It sends response 150 (data connection will open shortly).
8. The client sends the LIST message.
9. Now the server responds with 125 and opens the data connection.
10. The server then sends the list of the files or directories (as a file) on the data connection. When the whole list (file) is sent, the server responds with 226 (closing data connection) over the control connection.
11. The client now has two choices. It can use the QUIT command to request the closing of the control connection or it can send another command to start another activity (and eventually open another data connection). In our example, the client sends a QUIT command.
12. After receiving the QUIT command, the server responds with 221 (service closing) and then closes the control connection.

### Example 21.2

The following shows an actual FTP session that parallels Example 21.1. The colored lines show the responses from the server control connection; the black lines show the commands sent by the client. The lines in white with black background show data transfer.

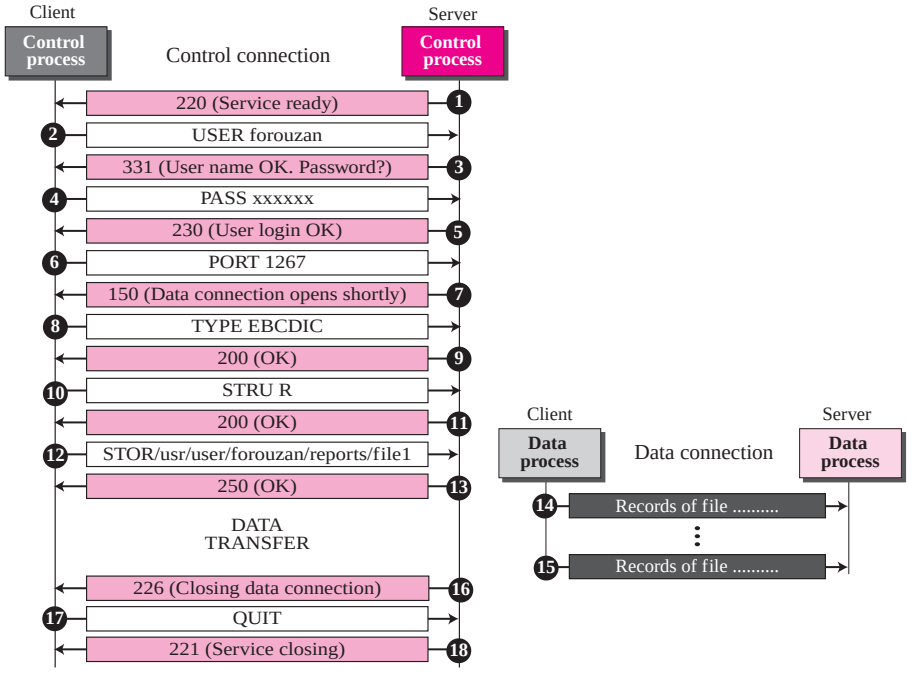
```
$ ftp voyager.deanza.fhda.edu
Connected to voyager.deanza.fhda.edu.
220 (vsFTPd 1.2.1)
530 Please login with USER and PASS.
Name (voyager.deanza.fhda.edu:forouzan): forouzan
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls reports
227 Entering Passive Mode (153,18,17,11,238,169)
150 Here comes the directory listing.
drwxr-xr-x  2   3027   411   4096  Sep 24  2002   business
drwxr-xr-x  2   3027   411   4096  Sep 24  2002   personal
drwxr-xr-x  2   3027   411   4096  Sep 24  2002   school
226 Directory send OK.
ftp> quit
221 Goodbye.
```

### Example 21.3

Figure 21.9 shows an example of how an image (binary) file is stored.

1. After the control connection to port 21 is created, the FTP server sends the 220 (service ready) response on the control connection.
2. The client sends the USER command.
3. The server responds with 331 (user name is OK, a password is required).
4. The client sends the PASS command.
5. The server responds with 230 (user login is OK).
6. The client issues a passive open on an ephemeral port for the data connection and sends the PORT command (over the control connection) to give this port number to the server.
7. The server does not open the connection at this time, but prepares itself for issuing an active open on the data connection between port 20 (server side) and the ephemeral port received from the client. It sends the response 150 (data connection will open shortly).
8. The client sends the TYPE command.
9. The server responds with the response 200 (command OK).
10. The client sends the STRU command.
11. The server responds with 200 (command OK).
12. The client sends the STOR command.
13. The server opens the data connection and sends the response 250.
14. The client sends the file on the data connection. After the entire file is sent, the data connection is closed. Closing the data connection means end-of-file.

**Figure 21.9** *Example 21.3*




15. The server sends the response 226 on the control connection.
16. The client sends the QUIT command or uses other commands to open another data connection for transferring another file. In our example, the QUIT command is sent.
17. The server responds with 221 (service closing) and it closes the control connection.

## Anonymous FTP

To use FTP, a user needs an account (user name) and a password on the remote server. Some sites have a set of files available for public access. To access these files, a user does not need to have an account or password. Instead, the user can use *anonymous* as the user name and *guest* as the password.

User access to the system is very limited. Some sites allow anonymous users only a subset of commands. For example, most sites allow the user to copy some files, but do not allow navigation through the directories.

### Example 21.4

We show an example of anonymous FTP. We assume that some public data are available at internic.net.

```
$ ftp internic.net
Connected to internic.net
220 Server ready
Name: anonymous
331 Guest login OK, send "guest" as password
Password: guest
ftp > pwd
257 '/' is current directory
ftp > ls
200 OK
150 Opening ASCII mode
bin
. . .
. . .
. . .
ftp > close
221 Goodbye
ftp > quit
```

### Security for FTP

The FTP protocol was designed when the security was not a big issue. Although FTP requires a password, the password is sent in plaintext (unencrypted), which means it can be intercepted and used by an attacker. The data transfer connection also transfers data in plaintext, which is insecure. To be secure, one can add a Secure Socket Layer (see Chapter 30) between the FTP application layer and the TCP layer. In this case FTP is called SSL-FTP.

### The sftp Program

Another way to transfer files using a secure channel is to use another independent protocol called **sftp** (secure file transfer protocol). This is actually a program in Unix called sftp that is part of the SSH protocol (see Chapter 20). When SSH has established a secure connection between an SSH client and an SSH server, one of the application programs that can use this connection (multiplexing) is sftp. In other words, sftp is part of the application component of the SSH. The sftp program is an interactive program that can work like FTP and uses a set of interface commands to transfer files between the SSH client and SSH server.

## 21.2 TFTP

There are occasions when we need to simply copy a file without the need for all of the features of the FTP protocol. For example, when a diskless workstation or a router is booted, we need to download the bootstrap and configuration files. Here we do not need all of the sophistication provided in FTP. We just need a protocol that quickly copies the files.

**Trivial File Transfer Protocol (TFTP)** is designed for these types of file transfer. It is so simple that the software package can fit into the read-only memory of a diskless workstation. It can be used at bootstrap time. The reason that it fits on ROM is that it

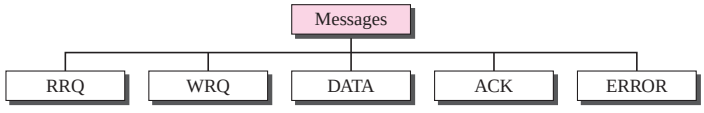
requires only basic IP and UDP. However, there is no security for TFTP. TFTP can read or write a file for the client. *Reading* means copying a file from the server site to the client site. *Writing* means copying a file from the client site to the server site.

**TFTP uses the services of UDP on the well-known port 69.**

## Messages

There are five types of TFTP messages, RRQ, WRQ, DATA, ACK, and ERROR, as shown in Figure 21.10.

**Figure 21.10** *Message categories*




### *RRQ*

The read request (RRQ) message is used by the client to establish a connection for reading data from the server. Its format is shown in Figure 21.11.

**Figure 21.11** *RRQ format*


The RRQ message fields are as follows:

- **OpCode.** The first field is a 2-byte operation code. The value is 1 for the RRQ message.
- **File name.** The next field is a variable-size string (encoded in ASCII) that defines the name of the file. Since the file name varies in length, termination is signaled by a 1-byte field of 0s.
- **Mode.** The next field is another variable-size string defining the transfer mode. The mode field is terminated by another 1-byte field of 0s. The mode can be one of two strings: "netascii" (for an ASCII file) or "octet" (for a binary file). The file name and mode fields can be in upper- or lowercase, or a combination of both.

### *WRQ*

The write request (WRQ) message is used by the client to establish a connection for writing data to the server. The format is the same as RRQ except that the OpCode is 2 (see Figure 21.12).

**Figure 21.12** *WRQ format*


### *DATA*

The data (DATA) message is used by the client or the server to send blocks of data. Its format is shown in Figure 21.13. The DATA message fields are as follows:

- **OpCode.** The first field is a 2-byte operation code. The value is 3 for the DATA message.

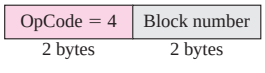
**Figure 21.13** *DATA format*


- **Block number.** This is a 2-byte field containing the block number. The sender of the data (client or server) uses this field for sequencing. All blocks are numbered sequentially starting with 1. The block number is necessary for acknowledgment as we will see shortly.
- **Data.** This block must be exactly 512 bytes in all DATA messages except the last block, which must be between 0 and 511 bytes. A non-512 byte block is used as a signal that the sender has sent all the data. In other words, it is used as an end-of-file indicator. If the data in the file happens to be an exact multiple of 512 bytes, the sender must send one extra block of zero bytes to show the end of transmission. Data can be transferred in either NVT ASCII (netascii) or binary octet (octet).

### *ACK*

The acknowledge (ACK) message is used by the client or server to acknowledge the receipt of a data block. The message is only 4 bytes long. Its format is shown in Figure 21.14.

**Figure 21.14** *ACK format*




The ACK message fields are as follows:

- **OpCode.** The first field is a 2-byte operation code. The value is 4 for the ACK message.
- **Block number.** The next field is a 2-byte field containing the number of the block received.

The ACK message can also be a response to a WRQ. It is sent by the server to indicate that it is ready to receive data from the client. In this case the value of the block number field is 0. An example of an ACK message is given in a later section.

### *ERROR*

The ERROR message is used by the client or the server when a connection cannot be established or when there is a problem during data transmission. It can be sent as a negative response to RRQ or WRQ. It can also be used if the next block cannot be transferred during the actual data transfer phase. The error message is not used to declare a damaged or duplicated message. These problems are resolved by error-control mechanisms discussed later in this chapter. The format of the ERROR message is shown in Figure 21.15.

**Figure 21.15** *ERROR format*


The ERROR message fields are as follows:

- **OpCode.** The first field is a 2-byte operation code. The value is 5 for the ERROR message.
- **Error number.** This 2-byte field defines the type of error. Table 21.8 shows the error numbers and their corresponding meanings.

**Table 21.8** *Error numbers and their meanings*

| *Number* | *Meaning* | *Number* | *Meaning* |
|---|---|---|---|
| 0 | Not defined | 5 | Unknown port number |
| 1 | File not found | 6 | File already exists |
| 2 | Access violation | 7 | No such user |
| 3 | Disk full or quota exceeded | | |
| 4 | Illegal operation | | |

- **Error data.** This variable-byte field contains the textual error data and is terminated by a 1-byte field of 0s.

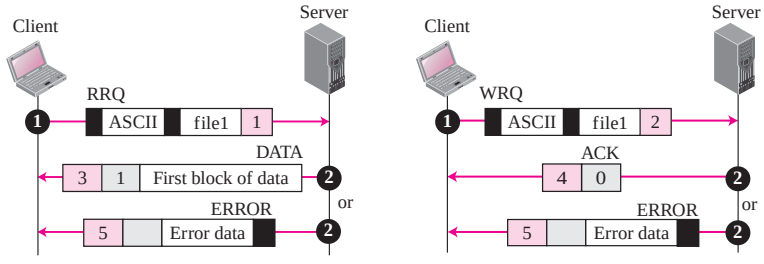
## Connection

TFTP uses UDP services. Because there is no provision for connection establishment and termination in UDP, UDP transfers each block of data encapsulated in an independent user datagram. In TFTP, however, we do not want to transfer only one block of data; we do not want to transfer the file as independent blocks either. We need connections for the blocks of data being transferred if they all belong to the same file. TFTP uses RRQ, WRQ, ACK, and ERROR messages to establish connection. It uses the DATA message with a block of data of fewer than 512 bytes (0–511) to terminate connection.

### *Connection Establishment*

**Connection establishment** for reading files is different from connection establishment for writing files (see Figure 21.16).

**Figure 21.16** *Connection establishment*




- **Reading.** To establish a connection for **reading,** the TFTP client sends the RRQ message. The name of the file and the transmission mode is defined in this message. If the server can transfer the file, it responds positively with a DATA message containing the first block of data. If there is a problem, such as difficulty in opening the file or permission restriction, the server responds negatively by sending an ERROR message.
- **Writing.** To establish a connection for **writing,** the TFTP client uses the WRQ message. The name of the file and the transmission mode is defined in this message. If the server can accept a copy of the file, it responds positively with an ACK message using a value of 0 for the block number. If there is any problem, the server responds negatively by sending an ERROR message.

### *Connection Termination*

After the entire file is transferred, the connection must be terminated. As mentioned previously, TFTP does not have a special message for termination. Termination is accomplished by sending the last block of data, which is less than 512 bytes.

## Data Transfer

The data transfer phase occurs between connection establishment and termination. TFTP uses the services of UDP, which is unreliable.

The file is divided into blocks of data, in which each block except the last one is exactly 512 bytes. The last block must be between 0 and 511 bytes. TFTP can transfer data in ASCII or binary format.

UDP does not have any mechanism for flow and error control. TFTP has to create a flow- and error-control mechanism to transfer a file made of continuous blocks of data.

### *Flow Control*

TFTP sends a block of data using the DATA message and waits for an ACK message. If the sender receives an acknowledgment before the time-out, it sends the next block. Thus, **flow control** is achieved by numbering the data blocks and waiting for an ACK before the next data block is sent.

**Retrieve a File** When the client wants to retrieve (read) a file, it sends the RRQ message. The server responds with a DATA message sending the first block of data (if there is no problem) with a block number of 1.

**Store a File** When the client wants to store (write) a file, it sends the WRQ message. The server responds with an ACK message (if there is no problem) using 0 for the block number. After receiving this acknowledgment, the client sends the first data block with a block number of 1.

### *Error Control*

The TFTP error-control mechanism is different from those of other protocols. It is *symmetric*, which means that the sender and the receiver both use time-outs. The sender uses a time-out for data messages; the receiver uses a time-out for acknowledgment messages. If a data message is lost, the sender retransmits it after time-out expiration. If an acknowledgment is lost, the receiver retransmits it after time-out expiration. This guarantees a smooth operation.

**Error control** is needed in four situations: a damaged message, a lost message, a lost acknowledgment, or a duplicated message.

**Damaged Message** There is no negative acknowledgment. If a block of data is damaged, it is detected by the receiver and the block is discarded. The sender waits for the acknowledgment and does not receive it within the time-out period. The block is then sent again. Note that there is no checksum field in the DATA message of TFTP. The only way the receiver can detect data corruption is through the checksum field of the UDP user datagram.

**Lost Message** If a block is lost, it never reaches the receiver and no acknowledgment is sent. The sender resends the block after the time-out.

**Lost Acknowledgment** If an acknowledgment is lost, we can have two situations. If the timer of the receiver matures before the timer of the sender, the receiver retransmits the acknowledgment; otherwise, the sender retransmits the data.

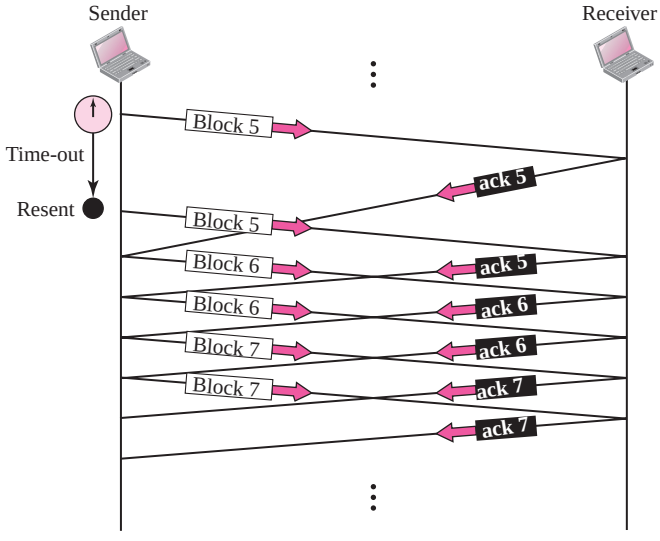
**Duplicate Message** Duplication of blocks can be detected by the receiver through block number. If a block is duplicated, it is simply discarded by the receiver.

### *Sorcerer's Apprentice Bug*

Although the flow- and error-control mechanism is symmetric in TFTP, it can lead to a problem known as the **sorcerer's apprentice bug,** named for the cartoon character who inadvertently conjures up a mop that continuously replicates itself. This will happen if the ACK message for a packet is not lost but delayed. In this situation, every succeeding block is sent twice and every succeeding acknowledgment is received twice.

Figure 21.17 shows this situation. The acknowledgment for the fifth block is delayed. After the time-out expiration, the sender retransmits the fifth block, which will be acknowledged by the receiver again. The sender receives two acknowledgments for the fifth block, which triggers it to send the sixth block twice. The receiver receives the sixth block twice and again sends two acknowledgments, which results in sending the seventh block twice. And so on.

**Figure 21.17** *Sorcerer's apprentice bug*




## UDP Ports

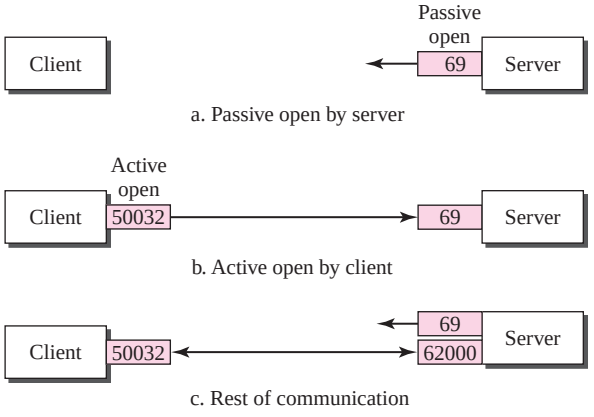
When a process uses the services of UDP, the server process issues a passive open on the well-known port and waits for the client process to issue an active open on an ephemeral port. After the connection is established, the client and server communicate using these two ports.

TFTP follows a different set of steps because the communication between a client TFTP and a server TFTP can be quite lengthy (seconds or even minutes). If a TFTP server uses the well-known port 69 to communicate with a single client, no other clients can use these services during that time. The solution to this problem, as shown in Figure 21.18, is to use the well-known port for the initial connection and an ephemeral port for the remaining communication.

The steps are as follows:

1. The server passively opens the connection using the well-known port 69.
2. A client actively opens a connection using an ephemeral port for the source port and the well-known port 69 for the destination port. This is done through the RRQ message or the WRQ message.

**Figure 21.18** *UDP port numbers used by TFTP*




3. The server actively opens a connection using a new ephemeral port for the source port and uses the ephemeral port received from the client as the destination port. It sends the DATA or ACK or ERROR message using these ports. This frees the well-known port (69) for use by other clients. When the client receives the first message from the server, it uses its own ephemeral port and the ephemeral port sent by the server for future communication.

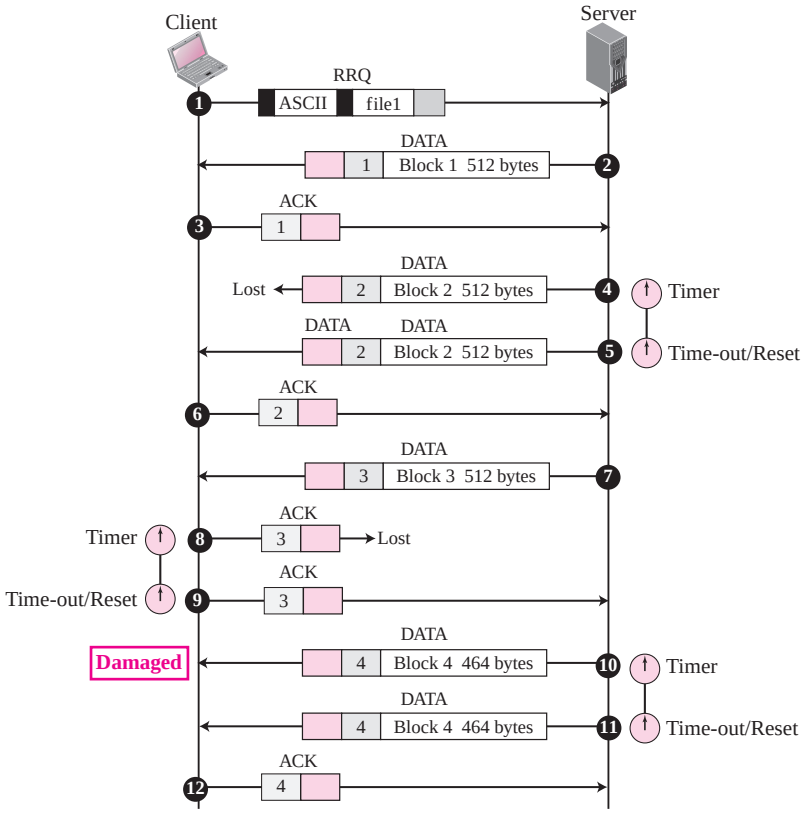
## TFTP Example

Figure 21.19 shows an example of a TFTP transmission. The client wants to retrieve a copy of the contents of a 2,000-byte file called *file1*. The client sends an RRQ message. The server sends the first block, carrying the first 512 bytes, which is received intact and acknowledged. These two messages are the connection establishment. The second block, carrying the second 512 bytes, is lost. After the time-out, the server retransmits the block, which is received. The third block, carrying the third 512 bytes, is received intact, but the acknowledgment is lost. After the time-out, the receiver retransmits the acknowledgment. The last block, carrying the remaining 464 bytes, is received damaged, so the client simply discards it. After the time-out, the server retransmits the block. This message is considered the connection termination because the block carries fewer than 512 bytes.

## TFTP Options

An extension to the TFTP protocol that allows the appending of options to the RRQ and WRQ messages has been proposed. The options are mainly used to negotiate the size of the block and possibly the initial sequence number. Without the options the size of a block is 512 bytes except for the last block. The negotiation can define a size of block to be any number of bytes so long as the message can be encapsulated in a UDP user datagram.

**Figure 21.19** *TFTP example*




A new type of message, option acknowledgment (OACK), to let the other party accept or reject the options, has also been proposed.

## Security

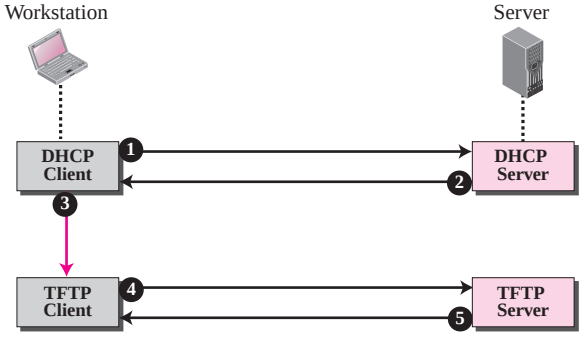
One important point about TFTP is that there is no provision for security: There is no user identification or password. Today, however, precautions must be taken to prevent hackers from accessing files. One security measure is to limit the access of TFTP to noncritical files. One way to achieve minimal security is to implement security in the router close to a TFTP server, which would allow only certain hosts to access the server.

## Applications

TFTP is very useful for basic file transfer where security is not a big issue. It can be used to initialize devices such as bridges or routers. Its main application is in conjunction with the DHCP. TFTP requires only a small amount of memory and uses only the services of UDP and IP. It can easily be configured in ROM (or PROM). When the station is

powered on, TFTP will be connected to a server and can download the configuration files from there. Figure 21.20 shows the idea. The powered-on station uses the DHCP client to get the name of the configuration file from the DHCP server. The station then passes the name of the file to the TFTP client to get the contents of the configuration file from the TFTP server.

**Figure 21.20** *Use of TFTP with DHCP*




## 21.3 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books and RFCs give an easy but thorough coverage of FTP and TFTP including [Com 06], [Mir 07], and [Ste 94].

### RFCs

Several RFCs show updates on FTP, including RFC 959, RFC 2577, RFC 2585. More information about TFTP can be found in RFC 906, RFC 1350, RFC 2347, RFC 2348, and RFC 2349.

## 21.4 KEY TERMS

anonymous FTP
ASCII file
block mode
compressed mode
connection establishment
control connection
data connection
EBCDIC file
file structure
File Transfer Protocol (FTP)
flow control
image file

reading
record structure
sorcerer's apprentice bug
sftp
stream mode
Trivial File Transfer Protocol (TFTP)
writing

## 21.5 SUMMARY

- File Transfer Protocol (FTP) is a TCP/IP client-server application for copying files from one host to another. FTP requires two connections for data transfer: a control connection and a data connection. FTP employs NVT ASCII for communication between dissimilar systems. Prior to the actual transfer of files, the file type, data structure, and transmission mode are defined by the client through the control connection.
- There are six classes of commands sent by the client to establish communication with the server: access commands, file management commands, data formatting commands, port defining commands, file transferring commands, and miscellaneous commands. There are three types of file transfer: server-to-client file transfer, client-to-server file transfer, transfer of list of directories.
- Transferring files with FTP is not secure. One solution to provide security is to add a Secure Socket Layer (SSL) between the FTP application layer and the TCP layer. Another solution is to use a completely independent file transfer application called sftp that is one of the application in SSH protocol.
- Trivial File Transfer Protocol (TFTP) is a simple file transfer protocol without the complexities and sophistication of FTP. A client uses the services of TFTP to retrieve a copy of a file or send a copy of a file to a server. There are five types of TFTP messages: RRQ, WRQ, DATA, ACK, and ERROR. TFTP can be used in conjunction with DHCP to initialize devices by downloading configuration files.
- In TFTP, error control is needed in four situations: a damaged message, a lost message, a lost acknowledgment, or a duplicated message. The sorcerer's apprentice bug is the duplication of both acknowledgments and data messages caused by TFTP's flow- and error-control mechanism.

## 21.6 PRACTICE SET

### Exercises

1. What do you think would happen if the control connection is accidentally severed during an FTP transfer?
2. Explain why the client issues an active open for the control connection and a passive open for the data connection.
3. Why should there be limitations on anonymous FTP? What could an unscrupulous user do?

4. Explain why FTP does not have a message format.
5. Show a TCP segment carrying one of the FTP commands.
6. Show a TCP segment carrying one of the FTP responses.
7. Show a TCP segment carrying FTP data.
8. Explain what will happen if the file in Example 21.2 already exists.
9. Redo Example 21.1 using the PASV command instead of the PORT command.
10. Redo Example 21.2 using the STOU command instead of the STOR command to store a file with a unique name. What happens if a file already exists with the same name?
11. Redo Example 21.2 using the RETR command instead of the STOR command to retrieve a file.
12. Give an example of the use of the HELP command.
13. Give an example of the use of the NOOP command.
14. Give an example of the use of the SYST command.
15. A user wants to make a directory called *Jan* under the directory */usr/usrs/letters*. The host is called "*mcGraw.com*.". Show all of the commands and responses using Examples 21.1 and 21.2 as a guide.
16. A user wants to move to the parent of its current directory. The host is called "*mcGraw.com*.". Show all of the commands and responses using Examples 21.1 and 21.3 as a guide.
17. A user wants to move a file named *file1* from */usr/usrs/report* directory to */usr/usrs/letters* directory. The host is called "*mcGraw.com*.". Show all the commands and responses using Examples 21.1 and 21.2 as a guide.
18. A user wants to retrieve an EBCDIC file named *file1* from */usr/usrs/report* directory. The host is called "*mcGraw.com*.". The file is so large that the user wants to compress it before transferring. Show all the commands and responses using Examples 21.1 and 21.2 as a guide.
19. Why do we need an RRQ or WRQ message in TFTP but not in FTP?
20. Show the encapsulation of an RRQ message in a UDP user datagram. Assume the file name is "Report" and the mode is ASCII. What is the size of the UDP datagram?
21. Show the encapsulation of a WRQ message in a UDP user datagram. Assume the file name is "Report" and the mode is ASCII. What is the size of the UDP datagram?
22. Show the encapsulation of a TFTP data message, carrying block number 7, in a UDP user datagram. What is the total size of the user datagram?
23. Host A uses TFTP to read 2,150 bytes of data from host B.
    a. Show all the TFTP commands including commands needed for connection establishment and termination. Assume no error.
    b. Show all the user datagrams exchanged between the two hosts.
24. Redo Exercise 23 but assume the second block is in error. Show also all the user datagrams exchanged between the two hosts.

## Research Activities

**25.** Find how routers can use security for TFTP.

**26.** Use UNIX or Windows to find all commands used in FTP.

**27.** Use UNIX or Windows to find all commands used in TFTP.

**28.** Find the format of the proposed OACK message.

**29.** Find the types of options proposed to be appended to the RRQ and WRQ messages.

CHAPTER

# 22

# World Wide Web and HTTP

The **World Wide Web (WWW)** is a repository of information linked together from points all over the world. The WWW has a unique combination of flexibility, portability, and user-friendly features that distinguish it from other services provided by the Internet. The WWW project was initiated by CERN (European Laboratory for Particle Physics) to create a system to handle distributed resources necessary for scientific research. In this chapter we first discuss issues related to the Web. We then discuss a protocol, HTTP, that is used to retrieve information from the Web.

## OBJECTIVES

*The chapter has several objectives:*

- To discuss the architecture of WWW and describe the concepts of hypertext and hypermedia.
- To describe Web clients and Web servers and their components.
- To define URL as a tool to identify a Web server.
- To introduce three different Web documents: static document, dynamic document, and active document.
- To discuss HTTP and its transactions.
- To define and list the fields in a request message.
- To define and list the fields in a response message.
- To define nonpersistent and persistent connections in HTTP.
- To introduce cookies and their applications in HTTP.
- To discuss Web caching, its application, and the method used to update the cache.

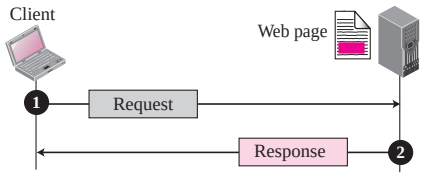
## 22.1 ARCHITECTURE

The WWW today is a distributed client-server service, in which a client using a browser can access a service using a server. However, the service provided is distributed over many locations called *sites*. Each site holds one or more documents, referred to as Web pages. Each **Web page**, however, can contain some links to other Web pages in the same or other sites. In other words, a Web page can be simple or composite. A simple Web page has no link to other Web pages; a composite Web page has one or more links to other Web pages. Each Web page is a file with a name and address.

### Example 22.1

Assume we need to retrieve a Web page that contains the biography of a famous character with some pictures, which are embedded in the page itself. Since the pictures are not stored as separate files, the whole document is a simple Web page. It can be retrieved using one single request/response transaction, as shown in Figure 22.1.

**Figure 22.1** *Example 22.1*

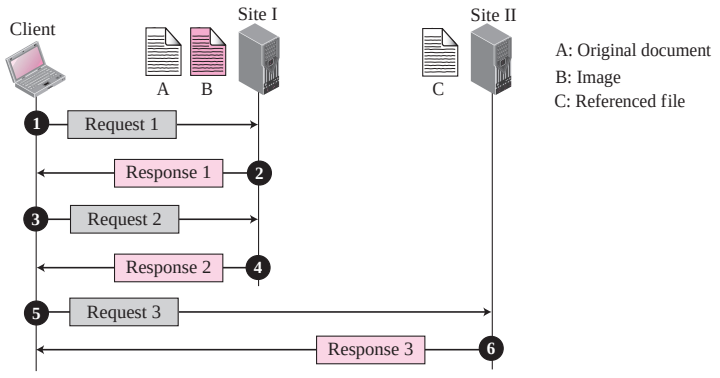



### Example 22.2

Now assume we need to retrieve a scientific document that contains one reference to another text file and one reference to a large image. Figure 22.2 shows the situation.

The main document and the image are stored in two separate files in the same site (file A and file B); the referenced text file is stored in another site (file C). Since we are dealing with three different files, we need three transactions if we want to see the whole document. The first transaction (request/response) retrieves a copy of the main document (file A), which has a reference (pointer) to the second and the third files. When a copy of the main document is retrieved and browsed, the user can click on the reference to the image to invoke the second transaction and retrieve a copy of the image (file B). If the user further needs to see the contents of the referenced text file, she can click on its reference (pointer) invoking the third transaction and retrieving a copy of the file C. Note that although files A and B both are stored in site I, they are independent files with different names and addresses. Two transactions are needed to retrieve them.

**Figure 22.2** *Example 22.2*




### Example 22.3

A very important point we need to remember is that file A, file B, and file C in Example 22.2 are independent Web pages, each with independent names and addresses. Although references to file B or C are included in file A, it does not mean that each of these files cannot be retrieved independently. A second user can retrieve file B with one transaction. A third user can retrieve file C with one transaction.

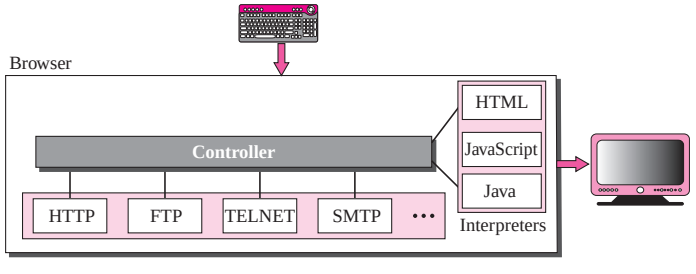
## Hypertext and Hypermedia

The three previous examples show the idea of **hypertext** and **hypermedia**. Hypertext means creating documents that refer to other documents. In a hypertext document, a part of text can be defined as a link to another document. When a hypertext is viewed with a browser, the link can be clicked to retrieve the other document. Hypermedia is a term applied to document that contains links to other textual document or documents containing graphics, video, or audio.

## Web Client (Browser)

A variety of vendors offer commercial **browsers** that interpret and display a Web document, and all of them use nearly the same architecture. Each browser usually consists of three parts: a controller, client protocol, and interpreters. (see Figure 22.3).

**Figure 22.3** *Browser*



The controller receives input from the keyboard or the mouse and uses the client programs to access the document. After the document has been accessed, the controller uses one of the interpreters to display the document on the screen. The client protocol can be one of the protocols described previously such as FTP, or TELNET, or HTTP (as discussed later in the chapter). The interpreter can be HTML, Java, or JavaScript, depending on the type of document. We discuss the use of these interpreters based on the document type later in the chapter. Some commercial browsers include Internet Explorer, Netscape Navigator, and Firefox.

## Web Server

The Web page is stored at the server. Each time a client request arrives, the corresponding document is sent to the client. To improve efficiency, servers normally store requested files in a cache in memory; memory is faster to access than disk. A server can also become more efficient through multithreading or multiprocessing. In this case, a server can answer more than one request at a time. Some popular Web servers include Apache and Microsoft Internet Information Server.

## Uniform Resource Locator (URL)

A client that wants to access a Web page needs the file name and the address. To facilitate the access of documents distributed throughout the world, HTTP uses locators. The **uniform resource locator (URL)** is a standard locator for specifying any kind of information on the Internet. The URL defines four things: protocol, host computer, port, and path (see Figure 22.4).

**Figure 22.4** *URL*


The *protocol* is the client-server application program used to retrieve the document. Many different protocols can retrieve a document; among them are Gopher, FTP, HTTP, News, and TELNET. The most common today is HTTP.

The **host** is the domain name of the computer on which the information is located. Web pages are usually stored in computers, and computers are given domain name aliases that usually begin with the characters "www". This is not mandatory, however, as the host can have any domain name.

The URL can optionally contain the port number of the server. If the *port* is included, it is inserted between the host and the path, and it is separated from the host by a colon.

**Path** is the pathname of the file where the information is located. Note that the path can itself contain slashes that, in the UNIX operating system, separate the directories from the subdirectories and files. In other words, the path defines the complete file name where the document is stored in the directory system.

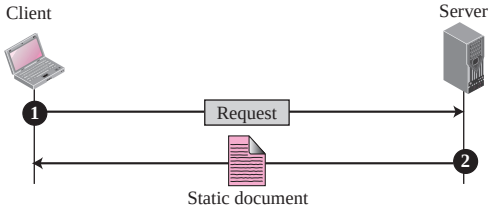
## 22.2 WEB DOCUMENTS

The documents in the WWW can be grouped into three broad categories: static, dynamic, and active. The category is based on the time the contents of the document are determined.

### Static Documents

**Static documents** are fixed-content documents that are created and stored in a server. The client can get a copy of the document only. In other words, the contents of the file are determined when the file is created, not when it is used. Of course, the contents in the server can be changed, but the user cannot change them. When a client accesses the document, a copy of the document is sent. The user can then use a browsing program to display the document (see Figure 22.5).

**Figure 22.5** *Static document*




Static documents are prepared using one of the several languages: **Hypertext Markup Language (HTML)**, **Extensible Markup Language (XML)**, **Extensible Style Language (XSL)**, and **Extended Hypertext Markup Language (XHTML)**. We discuss these languages in Appendix E.

**HTML, XML, XSL, and XHTML are discussed in Appendix E.**

### Dynamic Documents

A **dynamic document** is created by a Web server whenever a browser requests the document. When a request arrives, the Web server runs an application program or a script that creates the dynamic document. The server returns the output of the program or script as a response to the browser that requested the document. Because a fresh document is created for each request, the contents of a dynamic document may vary from one request to another. A very simple example of a dynamic document is the retrieval of the time and date from a server. Time and date are kinds of information that are dynamic in that they change from moment to moment. The client can ask the server to run a program such as the *date* program in UNIX and send the result of the program to the client.

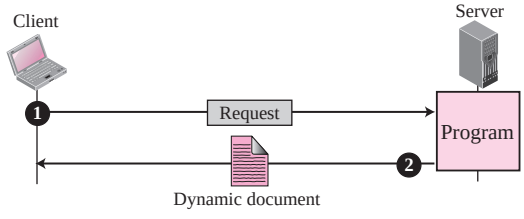
### *Common Gateway Interface (CGI)*

The **Common Gateway Interface (CGI)** is a technology that creates and handles dynamic documents. CGI is a set of standards that defines how a dynamic document is written, how data are input to the program, and how the output result is used.

CGI is not a new language; instead, it allows programmers to use any of several languages such as C, C++, Bourne Shell, Korn Shell, C Shell, Tcl, or Perl. The only thing that CGI defines is a set of rules and terms that the programmer must follow.

The term *common* in CGI indicates that the standard defines a set of rules that is common to any language or platform. The term *gateway* here means that a CGI program can be used to access other resources such as databases, graphic packages, and so on. The term *interface* here means that there is a set of predefined terms, variables, calls, and so on that can be used in any CGI program. A CGI program in its simplest form is code written in one of the languages supporting CGI. Any programmer who can encode a sequence of thoughts in a program and knows the syntax of one of the above-mentioned languages can write a simple CGI program. Figure 22.6 illustrates the steps in creating a dynamic program using CGI technology.

**Figure 22.6** *Dynamic document using CGI*




**Input** In traditional programming, when a program is executed, parameters can be passed to the program. Parameter passing allows the programmer to write a generic program that can be used in different situations. For example, a generic copy program can be written to copy any file to another. A user can use the program to copy a file named *x* to another file named *y* by passing *x* and *y* as parameters.

The input from a browser to a server is sent using a *form*. If the information in a form is small (such as a word), it can be appended to the URL after a question mark. For example, the following URL is carrying form information (23, a value):

**http://www.deanza/cgi-bin/prog.pl?23**

When the server receives the URL, it uses the part of the URL before the question mark to access the program to be run, and it interprets the part after the question mark (23) as the input sent by the client. It stores this string in a variable. When the CGI program is executed, it can access this value.

If the input from a browser is too long to fit in the query string, the browser can ask the server to send a form. The browser can then fill the form with the input data and send it to the server. The information in the form can be used as the input to the CGI program.

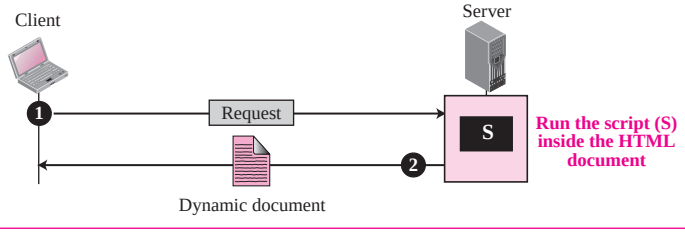
**Output** The whole idea of CGI is to execute a CGI program at the server site and send the output to the client (browser). The output is usually plain text or a text with HTML structures; however, the output can be a variety of other things. It can be graphics or binary data, a status code, instructions to the browser to cache the result, or instructions to the server to send an existing document instead of the actual output.

To let the client know about the type of document sent, a CGI program creates headers. As a matter of fact, the output of the CGI program always consists of two parts: a header and a body. The header is separated by a blank line from the body. This means any CGI program first creates the header, then a blank line, and then the body. Although the header and the blank line are not shown on the browser screen, the header is used by the browser to interpret the body.

### *Scripting Technologies for Dynamic Documents*

The problem with CGI technology is the inefficiency that results if part of the dynamic document that is to be created is fixed and not changing from request to request. For example, assume that we need to retrieve a list of spare parts, their availability, and prices for a specific car brand. Although the availability and prices vary from time to time, the name, description, and the picture of the parts are fixed. If we use CGI, the program must create an entire document each time a request is made. The solution is to create a file containing the fixed part of the document using HTML and embed a script, a source code, that can be run by the server to provide the varying availability and price section. Figure 22.7 shows the idea.

**Figure 22.7** *Dynamic document using server-site script*




A few technologies have been involved in creating dynamic documents using scripts. Among the most common are **Hypertext Preprocessor (PHP),** which uses the Perl language; **Java Server Pages (JSP),** which uses the Java language for scripting; **Active Server Pages (ASP),** a Microsoft product, which uses Visual Basic language for scripting; and **ColdFusion,** which embeds SQL database queries in the HTML document.

**Dynamic documents are sometimes referred to as server-site dynamic documents.**

## Active Documents

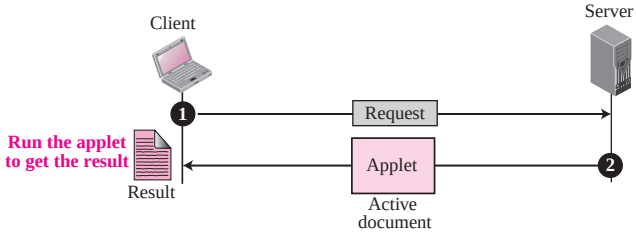
For many applications, we need a program or a script to be run at the client site. These are called **active documents.** For example, suppose we want to run a program that creates animated graphics on the screen or a program that interacts with the user. The program definitely needs to be run at the client site where the animation or interaction takes place. When a browser requests an active document, the server sends a copy of the document or a script. The document is then run at the client (browser) site.

### *Java Applets*

One way to create an active document is to use **Java applets. Java** is a combination of a high-level programming language, a run-time environment, and a class library that allows a programmer to write an active document (an applet) and a browser to run it. It can also be a stand-alone program that doesn't use a browser.

An **applet** is a program written in Java on the server. It is compiled and ready to be run. The document is in bytecode (binary) format. The client process (browser) creates an instance of this applet and runs it. A Java applet can be run by the browser in two ways. In the first method, the browser can directly request the Java applet program in the URL and receive the applet in binary form. In the second method, the browser can retrieve and run an HTML file that has embedded the address of the applet as a tag. Figure 22.8 shows how Java applets are used in the first method; the second is similar but needs two transactions.

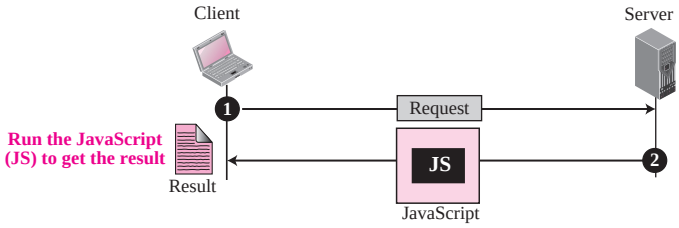
**Figure 22.8** *Active document using Java applet*




### *JavaScript*

The idea of scripts in dynamic documents can also be used for active documents. If the active part of the document is small, it can be written in a scripting language; then it can be interpreted and run by the client at the same time. The script is in source code (text) and not in binary form. The scripting technology used in this case is usually JavaScript. **JavaScript,** which bears a small resemblance to Java, is a very high level scripting language developed for this purpose. Figure 22.9 shows how JavaScript is used to create an active document.

**Figure 22.9** *Active document using client-site script*




**Active documents are sometimes referred to as client-site dynamic documents.**

## 22.3 HTTP

The **Hypertext Transfer Protocol (HTTP)** is a protocol used mainly to access data on the World Wide Web. HTTP functions like a combination of FTP (Chapter 21) and SMTP (Chapter 23). It is similar to FTP because it transfers files and uses the services of TCP. However, it is much simpler than FTP because it uses only one TCP connection. There is no separate control connection; only data are transferred between the client and the server.

HTTP is like SMTP because the data transferred between the client and the server look like SMTP messages. In addition, the format of the messages is controlled by MIME-like headers. Unlike SMTP, the HTTP messages are not destined to be read by humans; they are read and interpreted by the HTTP server and HTTP client (browser). SMTP messages are stored and forwarded, but HTTP messages are delivered immediately. The commands from the client to the server are embedded in a request message. The contents of the requested file or other information are embedded in a response message. HTTP uses the services of TCP on well-known port 80.

**HTTP uses the services of TCP on well-known port 80.**

### HTTP Transaction

Figure 22.10 illustrates the HTTP transaction between the client and server. Although HTTP uses the services of TCP, HTTP itself is a stateless protocol, which means that the server does not keep information about the client. The client initializes the transaction by sending a request. The server replies by sending a response.

#### *Request Message*

The format of the request is shown in Figure 22.11. A request message consists of a request line, a header, and sometimes a body.

**Request Line** The first line in a request message is called a **request line.** There are three fields in this line separated by some character delimiter as shown in Figure 22.11. The fields are called methods, URL, and Version. These three should be separated by a space character. At the end two characters, a carriage return followed by a line feed,

**Figure 22.10** *HTTP transaction*

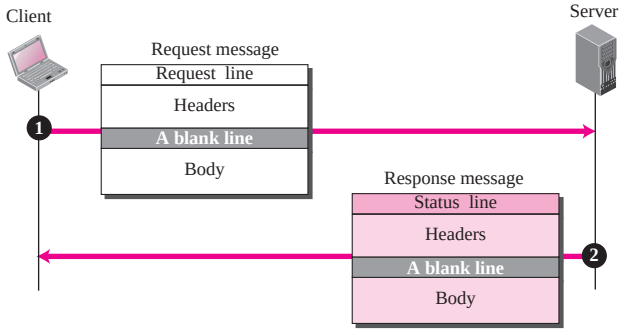
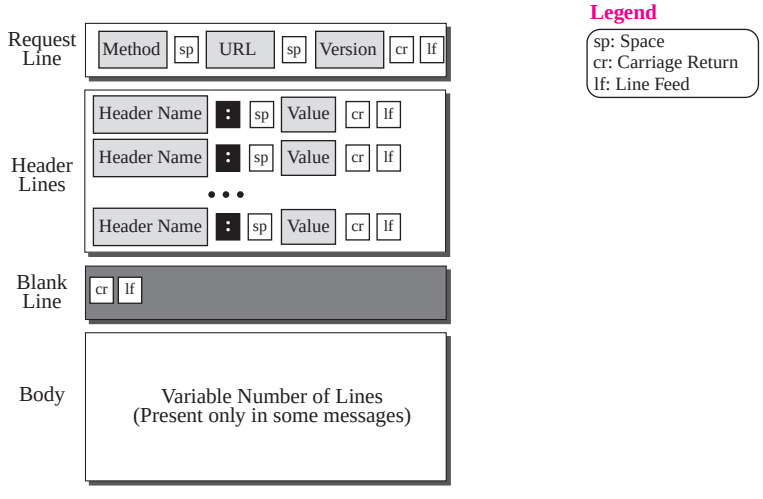



**Figure 22.11** *Format of the request message*




terminate the line. The method field defines the **request type.** In version 1.1 of HTTP, several methods are defined, as shown in Table 22.1.

**Table 22.1** *Methods*

| *Method* | *Action* |
|---|---|
| GET | Requests a document from the server |
| HEAD | Requests information about a document but not the document itself |
| POST | Sends some information from the client to the server |
| PUT | Sends a document from the server to the client |
| TRACE | Echoes the incoming request |
| CONNECT | Reserved |
| DELETE | Remove the Web page |
| OPTIONs | Enquires about available options |

The second field, URL, was discussed earlier in the chapter. It defines the address and name of corresponding Web page. The third field, version, gives the version of the protocol; the most current version of HTTP is 1.1.

**Header Lines In Request Message** After the request line, we can have zero or more **request header** lines. Each header line sends additional information from the client to the server. For example, the client can request that the document be sent in a special format. Each header line has a header name, a colon, a space, and a header value (see Figure 22.11). We will show some header lines in the examples at the end of this chapter. Table 22.2 shows some header names commonly used in a request. The value field defines the values associated with each header name. The list of values can be found in the corresponding RFCs.

**Table 22.2** *Request Header Names*

| *Header* | *Description* |
|---|---|
| User-agent | Identifies the client program |
| Accept | Shows the media format the client can accept |
| Accept-charset | Shows the character set the client can handle |
| Accept-encoding | Shows the encoding scheme the client can handle |
| Accept-language | Shows the language the client can accept |
| Authorization | Shows what permissions the client has |
| Host | Shows the host and port number of the client |
| Date | Shows the current date |
| Upgrade | Specifies the preferred communication protocol |
| Cookie | Returns the cookie to the server |
| If-Modified-Since | Returns the cookie to the server |

**Body In Request Message** The body can be present in a request message. Usually, it contains the comment to be sent.

### *Response Message*

The format of the response message is shown in Figure 22.12. A response message consists of a status line, header lines, a blank line and sometimes a body.

**Status Line** The first line in a response message is called the **status line**. There are three fields in this line separated by spaces and terminated by a carriage return and line feed. The first field defines the version of HTTP protocol, currently 1.1. The status code field defines the status of the request. It consists of three digits. Whereas the codes in the 100 range are only informational, the codes in the 200 range indicate a successful request. The codes in the 300 range redirect the client to another URL, and the codes in the 400 range indicate an error at the client site. Finally, the codes in the 500 range indicate an error at the server site. The status phrase explains the status code in text form. The possible values for the status code and status phrase are shown in Table 22.3.

**Header Lines In Response Message** After the status line, we can have zero or more **response header** lines. Each header line sends additional information from the server to the client. For example, the sender can send extra information about the document.

**Figure 22.12** *Format of the response message*

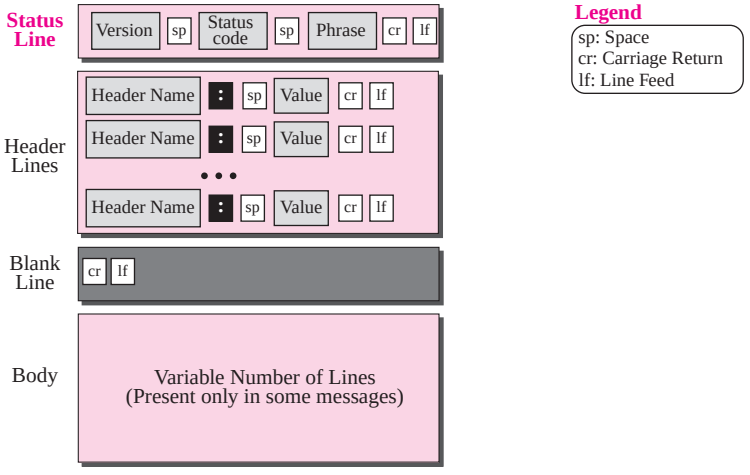



**Table 22.3** *Status Codes and Status Phrases*

| *Status Code* | *Status Phrase* | *Description* |
|---|---|---|
| | **Informational** | |
| 100 | Continue | The initial part of the request received, continue. |
| 101 | Switching | The server is complying to switch protocols. |
| | **Success** | |
| 200 | OK | The request is successful. |
| 201 | Created | A new URL is created. |
| 202 | Accepted | The request is accepted, but it is not immediately acted upon. |
| 204 | No content | There is no content in the body. |
| | **Redirection** | |
| 301 | Moved permanently | The requested URL is no longer used by the server. |
| 302 | Moved temporarily | The requested URL has moved temporarily. |
| 304 | Not modified | The document has not modified. |
| | **Client Error** | |
| 400 | Bad request | There is a syntax error in the request. |
| 401 | Unauthorized | The request lacks proper authorization. |
| 403 | Forbidden | Service is denied. |
| 404 | Not found | The document is not found. |
| 405 | Method not allowed | The method is not supported in this URL. |
| 406 | Not acceptable | The format requested is not acceptable. |
| | **Server Error** | |
| 500 | Internal server error | There is an error, such as a crash, at the server site. |
| 501 | Not implemented | The action requested cannot be performed. |
| 503 | Service unavailable | The service is temporarily unavailable. |

Each header line has a header name, a colon, a space, and a header value. We will show some header lines in the examples at the end of this chapter. Table 22.4 shows some header names commonly used in a response message.

**Table 22.4** *Response Header Names*

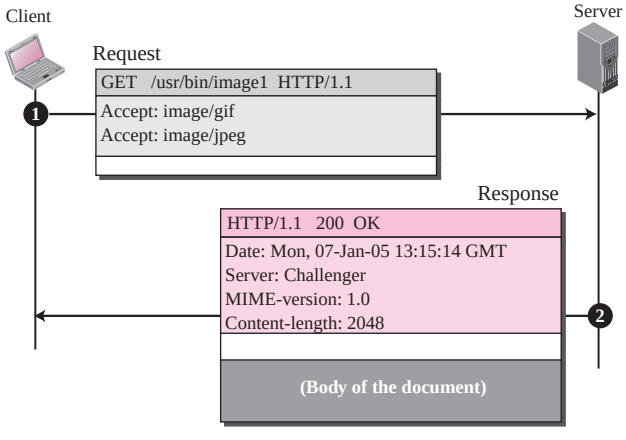
| *Header* | *Description* |
|---|---|
| Date | Shows the current date |
| Upgrade | Specifies the preferred communication protocol |
| Server | Gives information about the server |
| Set-Cookie | The server asks the client to save a cookie |
| Content-Encoding | Specifies the encoding scheme |
| Content-Language | Specifies the language |
| Content-Length | Shows the length of the document |
| Content-Type | Specifies the media type |
| Location | To ask the client to send the request to another site |
| Accept-Ranges | The server will accept the requested byte-ranges |
| Last-modified | Gives the date and time of the last change |

**Body** The body contains the document to be sent from the server to the client. The body is present unless the response is an error message.

### Example 22.4

This example retrieves a document (see Figure 22.13).

**Figure 22.13** *Example 22.4*

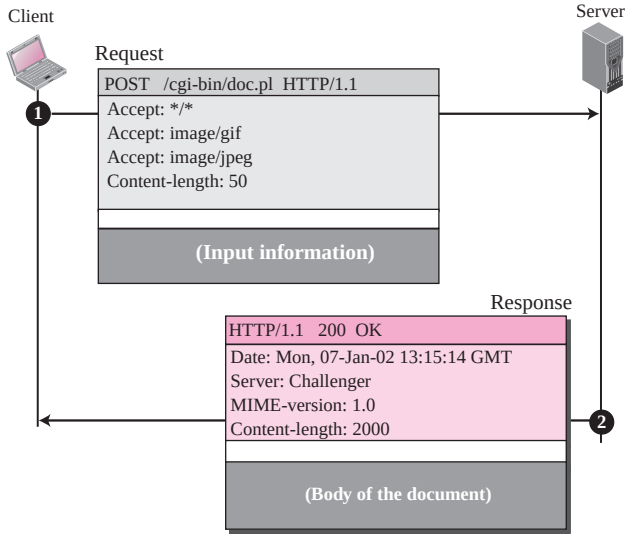



We use the GET method to retrieve an image with the path /usr/bin/image1. The request line shows the method (GET), the URL, and the HTTP version (1.1). The header has two lines that show that the client can accept images in the GIF or JPEG format. The request does not have a body. The response message contains the status line and four lines of header. The header lines define the date, server, MIME version, and length of the document. The body of the document follows the header.

### Example 22.5

In this example, the client wants to send data to the server. We use the POST method. The request line shows the method (POST), URL, and HTTP version (1.1). There are four lines of headers. The request body contains the input information. The response message contains the status line and four lines of headers. The created document, which is a CGI document, is included as the body (see Figure 22.14).

**Figure 22.14** *Example 22.5*




### Example 22.6

HTTP uses ASCII characters. The following shows how a client can directly connect to a server using TELNET, which logs into port 80. The first three lines shows that the connection is successful. We then type three lines. The first shows the request line (GET method), the second is the header (defining the host), the third is a blank terminating the request. The server response is seven lines starting with the status line. The blank line at the end terminates the server response. The file of 14,230 lines is received after the blank line (not shown here). The last line is the output by the client.

```
$telnet www.mhhe.com 80
Trying 198.45.24.104...
Connected to www.mhhe.com (198.45.24.104).
Escape character is '^]'.
GET /engcs/compsci/forouzan HTTP/1.1
From: forouzanbehrouz@fhda.edu
HTTP/1.1 200 OK
Date: Thu, 28 Oct 2004 16:27:46 GMT
Server: Apache/1.3.9 (Unix) ApacheJServ/1.1.2 PHP/4.1.2 PHP/3.0.18
MIME-version:1.0
```

```
Content-Type: text/html
Last-modified: Friday, 15-Oct-04 02:11:31 GMT
Content-length: 14230

Connection closed by foreign host.
```

## Conditional Request

A client can add a condition in its request. In this case, the server will send the requested Web page if the condition is met or inform the client otherwise. One of the most common conditions imposed by the client is the time and date the Web page is modified. The client can send the header line *If-Modified-Since* to the request to tell the server that it needs the page if it is modified after a certain point of time.

### Example 22.7

The following shows how a client imposes the modification data and time condition on a request.

```
GET http://www.commonServer.com/information/file1 HTTP/1.1
If-Modified-Since: Thu, Sept 04 00:00:00 GMT
```

The status line in the responds shows the file is not modified after the defined point of time. The body of the response message is also empty.

```
HTTP/1.1 304 Not Modified
Date: Sat, Sept 06 08 16:22:46 GMT
Server: commonServer.com

(Empty Body)
```

## Persistence

HTTP, prior to version 1.1, specified a nonpersistent connection, while a persistent connection is the default in version 1.1.

### *Nonpersistent Connection*

In a **nonpersistent connection,** one TCP connection is made for each request/response. The following lists the steps in this strategy:

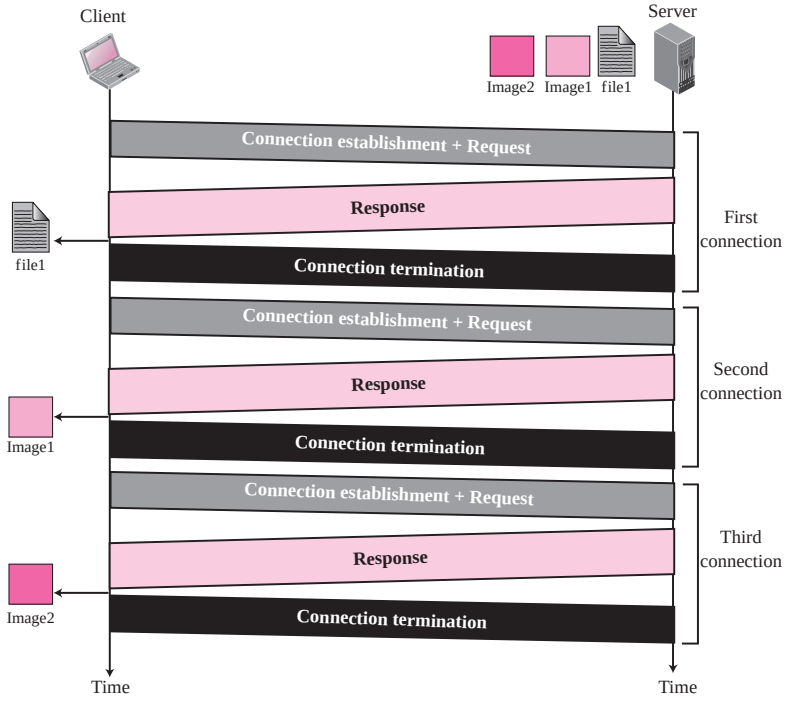
1. The client opens a TCP connection and sends a request.
2. The server sends the response and closes the connection.
3. The client reads the data until it encounters an end-of-file marker; it then closes the connection.

In this strategy, if a file contains link to $N$ different pictures in different files (all located on the same server), the connection must be opened and closed $N + 1$ times. The nonpersistent strategy imposes high overhead on the server because the server needs $N + 1$ different buffers and requires a slow start procedure each time a connection is opened.

### Example 22.8

Figure 22.15 shows an example of a nonpersistent connection. The client needs to access a file that contains two links to images. The text file and images are located on the same server.

**Figure 22.15** *Example 22.8*




### *Persistent Connection*

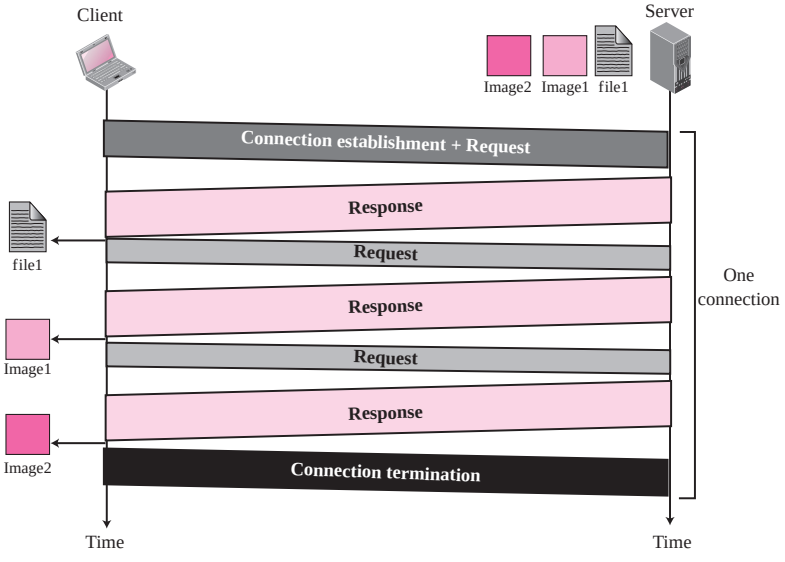
HTTP version 1.1 specifies a **persistent connection** by default. In a persistent connection, the server leaves the connection open for more requests after sending a response. The server can close the connection at the request of a client or if a time-out has been reached. The sender usually sends the length of the data with each response. However, there are some occasions when the sender does not know the length of the data. This is the case when a document is created dynamically or actively. In these cases, the server informs the client that the length is not known and closes the connection after sending the data so the client knows that the end of the data has been reached.

**HTTP version 1.1 specifies a persistent connection by default.**

### Example 22.9

Figure 22.16 shows the same scenario as Example 22.8, but using persistent connection.

**Figure 22.16** *Example 22.9*




## Cookies

The World Wide Web was originally designed as a stateless entity. A client sends a request; a server responds. Their relationship is over. The original design of WWW, retrieving publicly available documents, exactly fits this purpose. Today the Web has other functions; some are listed below:

- Websites are being used as *electronic stores* that allow users to browse through the store, select wanted items, put them in an electronic cart, and pay at the end with a credit card.
- Some websites need to allow access to *registered clients* only.
- Some websites are used as *portals:* The user selects the Web pages he wants to see.
- Some websites are just *advertising*.

For these purposes, the cookie mechanism was devised. We discussed the use of cookies at the transport layer in Chapter 15; we now discuss their use in Web pages.

### *Creating and Storing Cookies*

The creation and storing of cookies depend on the implementation; however, the principle is the same.

1. When a server receives a request from a client, it stores information about the client in a file or a string. The information may include the domain name of the client, the contents of the cookie (information the server has gathered about the client such as

name, registration number, and so on), a timestamp, and other information depending on the implementation.

2. The server includes the cookie in the response that it sends to the client.
3. When the client receives the response, the browser stores the cookie in the cookie directory, which is sorted by the domain server name.

### *Using Cookies*

When a client sends a request to a server, the browser looks in the cookie directory to see if it can find a cookie sent by that server. If found, the cookie is included in the request. When the server receives the request, it knows that this is an old client, not a new one. Note that the contents of the cookie are never read by the browser or disclosed to the user. It is a cookie *made* by the server and *eaten* by the server. Now let us see how a cookie is used for the four previously mentioned purposes:

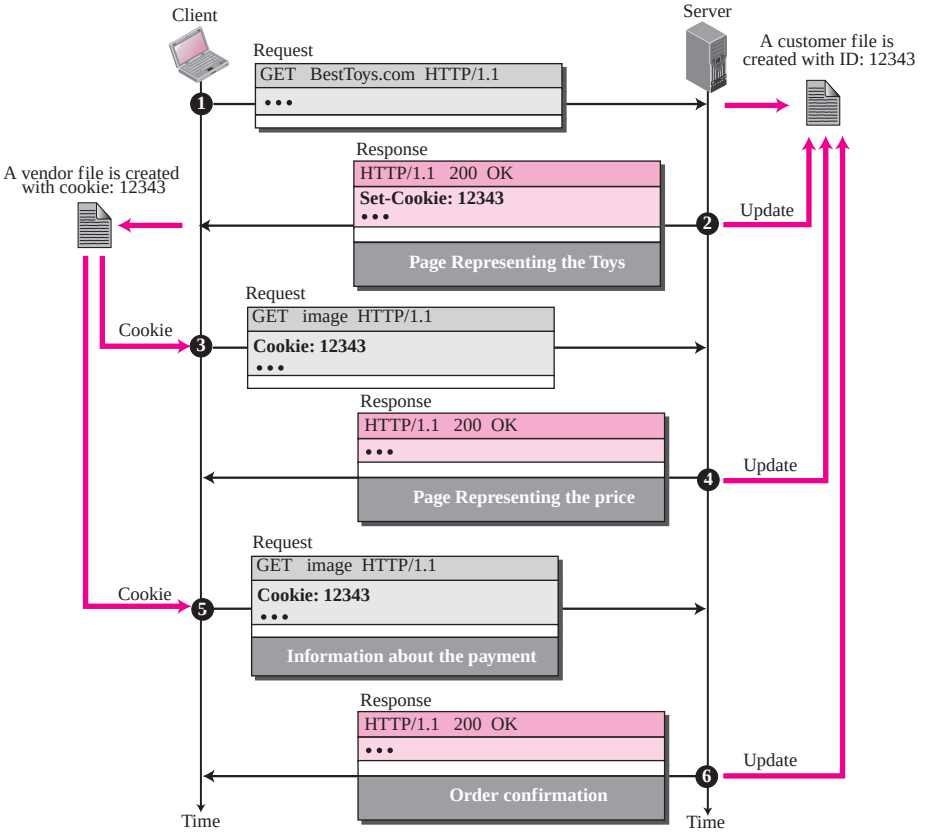
- An *electronic store* (e-commerce) can use a cookie for its client shoppers. When a client selects an item and inserts it into a cart, a cookie that contains information about the item, such as its number and unit price, is sent to the browser. If the client selects a second item, the cookie is updated with the new selection information. And so on. When the client finishes shopping and wants to check out, the last cookie is retrieved and the total charge is calculated.
- The site that restricts access to *registered clients* only sends a cookie to the client when the client registers for the first time. For any repeated access, only those clients that send the appropriate cookie are allowed.
- A Web *portal* uses the cookie in a similar way. When a user selects her favorite pages, a cookie is made and sent. If the site is accessed again, the cookie is sent to the server to show what the client is looking for.
- A cookie is also used by *advertising* agencies. An advertising agency can place banner ads on some main website that is often visited by users. The advertising agency supplies only a URL that gives the banner address instead of the banner itself. When a user visits the main website and clicks the icon of an advertised corporation, a request is sent to the advertising agency. The advertising agency sends the banner, a GIF file for example, but it also includes a cookie with the ID of the user. Any future use of the banners adds to the database that profiles the Web behavior of the user. The advertising agency has compiled the interests of the user and can sell this information to other parties. This use of cookies has made them very controversial. Hopefully, some new regulations will be devised to preserve the privacy of users.

### Example 22.10

Figure 22.17 shows a scenario in which an electronic store can benefit from the use of cookies. Assume a shopper wants to buy a toy from an electronic store named BestToys. The shopper browser (client) sends a request to the BestToys server. The server creates an empty shopping cart (a list) for the client and assigns an ID to the cart (for example, 12343). The server then sends a response message, which contains the images of all toys available with a

link under each toy that select the toy if it is being clicked. This response message also includes the Set-Cookie header line whose value is 12343. The client displays the images and store the cookie value in a file named BestToys. The cookie is not revealed to the shopper. Now the shopper selects one of the toys and clicks on it. The client sends a request, but includes the ID 12343 in the Cookie header line. Although the server may have been busy and has forgotten about this shopper, when it receives the request and check the header it finds the value 12343 as the cookie. The server knows that the customer is not new, it searches for a shopping cart with ID 12343. The shopping cart (list) is opened and the selected toy is inserted to the list. The server now sends another response to the shopper to tell her the total price and ask her to provide payment. The shopper provides information about her credit card and sends a new request with the ID 12343 as the cookie value. When the request arrives at the server, it again sees the ID 12343, and accepts the order and the payment and sends a confirmation in a response. Other information about the client, such as the credit card number, name, and address is stored in the server. If the shopper accesses the store sometime in the future, the client sends the cookie again; the store retrieves the file and has all information about the client.

**Figure 22.17** *Example 22.10*



## Web Caching: Proxy Server

HTTP supports **proxy servers.** A proxy server is a computer that keeps copies of responses to recent requests. The HTTP client sends a request to the proxy server. The proxy server checks its cache. If the response is not stored in the cache, the proxy server sends the request to the corresponding server. Incoming responses are sent to the proxy server and stored for future requests from other clients.

The proxy server reduces the load on the original server, decreases traffic, and improves latency. However, to use the proxy server, the client must be configured to access the proxy instead of the target server.

Note that the proxy server acts both as a server and client. When it receives a request from a client for which it has a response, it acts as a server and sends the response to the client. When it receives a request from a client for which it does not have a response, it first acts as a server and sends a request to the target server. When the response has been received, it acts again as a server and sends the response to the client.

### *Proxy Server Location*

The proxy servers are normally located at the client site. This means that we can have a hierarchy of proxy servers as shown below:

1. A client computer can also be used as a proxy server in a small capacity that stores responses to requests often invoked by the client.
2. In a company, a proxy server may be installed on the computer LAN to reduce the load going out of and coming into the LAN.
3. An ISP with many customers can install a proxy server to reduce the load going out of and coming into the ISP network.

### *Cache Update*

A very important question is how long a response should remain in the proxy server before being deleted and replaced. Several different strategies are used for this purpose. One solution is to store the list of sites whose information remains the same for a while. For example, a news agency may change its news page every morning. This means that a proxy server can get the news early in the morning and keep it until the next day. Another recommendation is to add some headers to show the last modification time of the information. The proxy server can then use the information in this header to guess how long the information would be valid. There are more recommendations for Web caching, but we leave them to more specific books on this subject.

## HTTP Security

The HTTP per se does not provide security. However, as we show in Chapter 30, HTTP can be run over the Secure Socket Layer (SSL). In this case, HTTP is referred to as HTTPS. HTTPS provides confidentiality, client and server authentication, and data integrity.

## 22.4 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give an easy but thorough coverage of HTTP, including [Com 06], [Mir 07], [Ste 94], and [Tan 03].

### RFCs

Several RFCs show updates on HTTP, including RFC 2068, and RFC 2109.

## 22.5 KEY TERMS

active document
Active Server Pages (ASP)
applet
browser
ColdFusion
Common Gateway Interface (CGI)
dynamic document
Extensible Markup Language (XML)
Extensisible Style Language (XSL)
host
hypermedia
hypertext
Hypertext Markup Language (HTML)
Hypertext Preprocessor (PHP)
Hypertext Transfer Protocol (HTTP)
Java
Java applet
JavaScript
Java Server Pages (JSP)
nonpersistent connection
path
persistent connection
proxy server
request header
request line
request type
response header
static document
status code
status line
uniform resource locator (URL)
Web page
World Wide Web (WWW)

## 22.6 SUMMARY

- The World Wide Web (WWW) is a repository of information linked together from points all over the world. Hypertext and hypermedia documents are linked to one another through pointers.
- The WWW architecture is made up of clients and servers. A client or a browser interprets and displays a Web document. A browser consists of a controller, client programs, and interpreters. A server stores Web pages.
- A Web document can be classified as static, dynamic, or active. A static document is one in which the contents are fixed and stored in a server. A dynamic Web

document is created by a server only at a browser request. An active document is a copy of a program retrieved by the client and run at the client site.

- The Hypertext Transfer Protocol (HTTP) is the main protocol used to access data on the World Wide Web (WWW). HTTP uses a TCP connection to transfer files. HTTP transactions are made of request and response messages.
- HTTP can be used in two modes: nonpersistent and persistent. The nonpersistent mode uses a new TCP connection for each transaction; the persistent mode uses only one connection. The default in the new version of HTTP is the persistent mode.
- HTTP can use cookies to keep the state of the transactions. The server sends a cookie that can be stored in the client and be retrieved later by the server.
- Web caching using proxy servers improves the efficiency of the HTTP. The proxy servers are installed in the client sites.
- A secure version of HTTP, HTTPS uses the services of a Secure Socket Layer (SSL) protocol to provide confidentiality, client and server authentication, and data integrity.

## 22.7 PRACTICE SET

### Exercises

1. Assume there is a server with the domain name *www.common.com*
   a. Show a request that retrieves the document /usr/users/doc/doc1. Use at least two general headers, two request headers, and one entity header.
   b. Show the response to part a for a successful request.
   c. Show the response to part a for a document that has permanently moved to /usr/deads/doc1.
   d. Show the response to part a if there is a syntax error in the request.
   e. Show the response to part a if the client is unauthorized to access the document.
2. Assume there is a server with the domain name *www.uncommon.com*
   a. Show a request that asks for information about a document at /bin/users/file. Use at least two general headers and one request header.
   b. Show the response to part a for a successful request.
3. Assume there is a server with the domain name *www.public.edu*
   a. Show the request to copy the file at location /bin/usr/bin/file1 to /bin/file1.
   b. Show the response to part a.
4. Assume there is a server with the domain name *www.bigBusiness.com*
   a. Show the request to delete the file at location /bin/file1.
   b. Show the response to part a.
5. Assume there is a server with the domain name *www.EveryOne.com*
   a. Show a request to store a file at location /bin/letter. The client identifies the types of documents it can accept.
   b. Show the response to part a. The response shows the age of the document as well as the date and time when the contents may change.

6. In Figure 22.5, find the actual number of TCP segments exchanged between the client and the server and draw a similar figure to show all segments.
7. In Figure 22.6, find the actual number of TCP segments exchanged between the client and the server and draw a similar figure to show all segments.
8. Draw a figure similar to Figure 22.17 to show the application of cookies in a scenario in which the server allows only the registered customer to access the server.
9. Draw a figure similar to Figure 22.17 to show the application of cookies in a Web portal.
10. Draw a figure similar to Figure 22.17 to show the application of cookies in a scenario in which the server uses cookies for advertisement.
11. Draw a diagram to show the use of a proxy server that is part of the client computer:
    a. Show the transactions between the client, proxy server, and the target server when the response is stored in the proxy server.
    b. Show the transactions between the client, proxy server, and the target server when the response is not stored in the proxy server.
12. Draw a diagram to show the use of a proxy server that is installed in the LAN where the client is:
    a. Show the transactions between the client, proxy server, and the target server when the response is stored in the proxy server.
    b. Show the transactions between the client, proxy server, and the target server when the response is not stored in the proxy server.
13. Draw a diagram to show the use of a proxy server that is installed in the ISP network:
    a. Show the transactions between the client, proxy server, and the target server when the response is stored in the proxy server.
    b. Show the transactions between the client, proxy server, and the target server when the response is not stored in the proxy server.

## Research Activities

14. Find out why IP addresses cannot replace cookies.
15. Find out about *Web mirroring*, in which caching is done at the server site instead of the client site.

CHAPTER

# 23

# *Electronic Mail: SMTP, POP, IMAP, and MIME*

One of the most popular Internet services is electronic mail (e-mail). The designers of the Internet probably never imagined the popularity of this application program. Its architecture consists of several components that we will discuss in this chapter.

At the beginning of the Internet era, the messages sent by electronic mail were short and consisted of text only; they let people exchange quick memos. Today, electronic mail is much more complex. It allows a message to include text, audio, and video. It also allows one message to be sent to one or more recipients.

In this chapter, we first study the general architecture of an e-mail system including the three main components: user agent, message transfer agent, and message access agent. We then describe the protocols that implement these components.

## OBJECTIVES

*The chapter has several objectives:*

- To explain the architecture of electronic mail using four scenarios.
- To explain the user agent (UA), services provided by it, and two types of user agents.
- To explain the mechanism of sending and receiving e-mails.
- To introduce the role of a message transfer agent and Simple Mail Transfer Protocol (SMTP) as the formal protocol that handles MTA.
- To explain e-mail transfer phases.
- To discuss two message access agents (MAAs): POP and IMAP.
- To discuss MIME as a set of software functions that transforms non-ASCII data to ASCII data and vice versa.
- To discuss the idea of Web-based e-mail.
- To explain the security of the e-mail system.

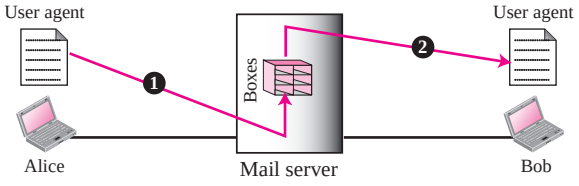
## 23.1 ARCHITECTURE

To explain the architecture of e-mail, we give four scenarios. We begin with the simplest situation and add complexity as we proceed. The fourth scenario is the most common in the exchange of e-mail.

### First Scenario

In the first scenario, the sender and the receiver of the e-mail are users (or application programs) on the same mail server; they are directly connected to a shared mail server. The administrator has created one mailbox for each user where the received messages are stored. A *mailbox* is part of a local hard drive, a special file with permission restrictions. Only the owner of the mailbox has access to it. When Alice needs to send a message to Bob, she runs a *user agent* (*UA*) program to prepare the message and store it in Bob's mailbox. The message has the sender and recipient mailbox addresses (names of files). Bob can retrieve and read the contents of his mailbox at his convenience using a user agent. Figure 23.1 shows the concept.

**Figure 23.1** *First scenario*



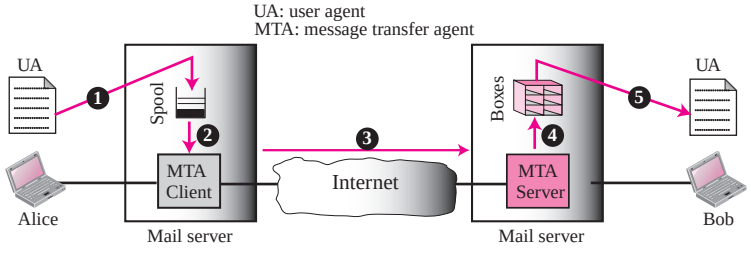

This is similar to the traditional memo exchange between employees in an office. There is a mail room where each employee has a mailbox with his or her name on it. When Alice needs to send a memo to Bob, she writes the memo and inserts it into Bob's mailbox. When Bob checks his mailbox, he finds Alice's memo and reads it.

**When the sender and the receiver of an e-mail are on the same mail server, we need only two user agents.**

## Second Scenario

In the second scenario, the sender and the receiver of the e-mail are users (or application programs) on two different mail servers. The message needs to be sent over the Internet. Here we need **user agents (UAs)** and **message transfer agents (MTAs)** as shown in Figure 23.2.

**Figure 23.2** *Second scenario*




Alice needs to use a user agent program to send her message to the mail server at her own site. The mail server at her site uses a queue (spool) to store messages waiting to be sent. Bob also needs a user agent program to retrieve messages stored in the mailbox of the system at his site. The message, however, needs to be sent through the Internet from Alice's site to Bob's site. Here two message transfer agents are needed: one client and one server. Like most client-server programs on the Internet, the server needs to run all of the time because it does not know when a client will ask for a connection. The client, on the other hand, can be triggered by the system when there is a message in the queue to be sent.

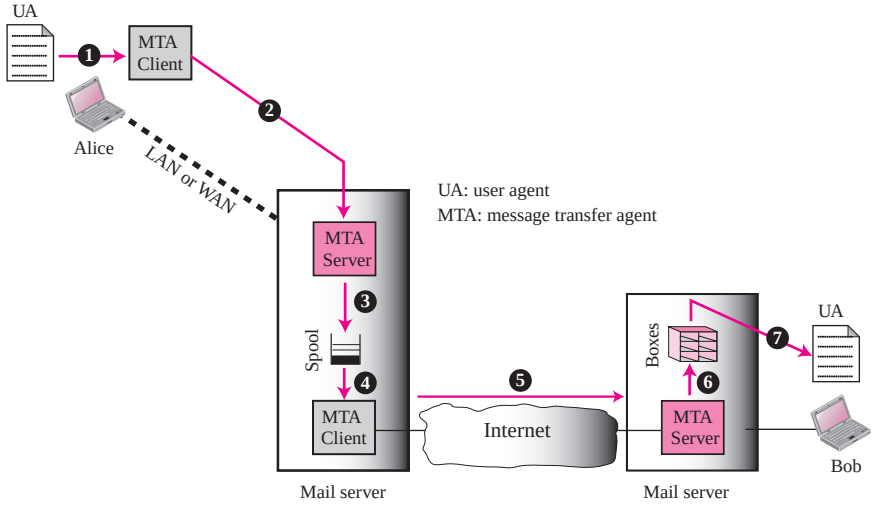
**When the sender and the receiver of an e-mail are on different mail servers, we need two UAs and a pair of MTAs (client and server).**

## Third Scenario

Figure 23.3 shows the third scenario. Bob, as in the second scenario, is directly connected to his mail server. Alice, however, is separated from her mail server. Alice is either connected to the mail server via a point-to-point WAN—such as a dial-up modem, a DSL, or a cable modem—or she is connected to a LAN in an organization that uses one mail server for handling e-mails; all users need to send their messages to this mail server.

Alice still needs a user agent to prepare her message. She then needs to send the message through the LAN or WAN. This can be done through a pair of message transfer agents (client and server). Whenever Alice has a message to send, she calls the user agent which, in turn, calls the MTA client. The MTA client establishes a connection with the MTA server on the system, which is running all the time. The system at Alice's site queues all messages received. It then uses an MTA client to send the messages to the system at Bob's site; the system receives the message and stores it in Bob's mailbox.

**Figure 23.3** *Third scenario*




At his convenience, Bob uses his user agent to retrieve the message and reads it. Note that we need two pairs of MTA client-server programs.

**When the sender is connected to the mail server via a LAN or a WAN, we need two UAs and two pairs of MTAs (client and server).**

## Fourth Scenario

In the fourth and most common scenario, Bob is also connected to his mail server by a WAN or a LAN. After the message has arrived at Bob's mail server, Bob needs to retrieve it. Here, we need another set of client-server agents, which we call **message access agents (MAAs).** Bob uses an MAA client to retrieve his messages. The client sends a request to the MAA server, which is running all the time, and requests the transfer of the messages. The situation is shown in Figure 23.4.

There are two important points we need to emphasize here. First, Bob cannot bypass the mail server and use the MTA server directly. To use the MTA server directly, Bob would need to run the MTA server all the time because he does not know when a message will arrive. This implies that Bob must keep his computer on all the time if he is connected to his system through a LAN. If he is connected through a WAN, he must keep the connection up all the time. Neither of these situations is feasible today.

Second, note that Bob needs another pair of client-server programs: message access programs. This is because an MTA client-server program is a *push* program: the client pushes the message to the server. Bob needs a *pull* program. The client needs to pull the message from the server. Figure 23.5 shows the difference.

**Figure 23.4** *Fourth scenario*

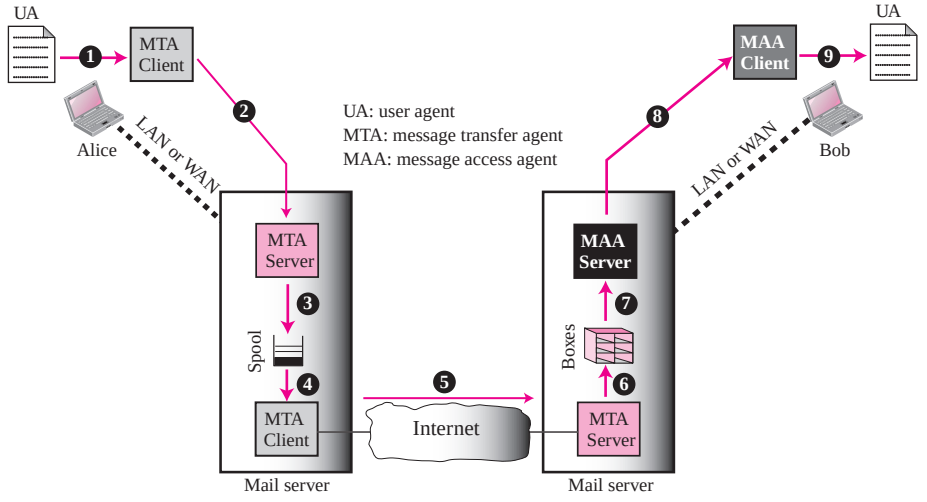



**Figure 23.5** *Push vs. pull*


a. Client pushes messages b. Client pulls messages

**When both sender and receiver are connected to the mail server via a LAN or a WAN, we need two UAs, two pairs of MTAs (client and server), and a pair of MAAs (client and server). This is the most common situation today.**

## 23.2 USER AGENT

The first component of an electronic mail system is the **user agent (UA).** It provides service to the user to make the process of sending and receiving a message easier.

### Services Provided by a User Agent

A user agent is a software package (program) that composes, reads, replies to, and forwards messages. It also handles local mailboxes on the user computers.

## User Agent Types

There are two types of user agents: command-driven and GUI-based. Command-driven user agents belong to the early days of electronic mail. They are still present as the underlying user agents in servers. A command-driven user agent normally accepts a one-character command from the keyboard to perform its task. For example, a user can type the character *r*, at the command prompt, to reply to the sender of the message, or type the character *R* to reply to the sender and all recipients.

**Some examples of command-driven user agents are *mail, pine,* and *elm*.**

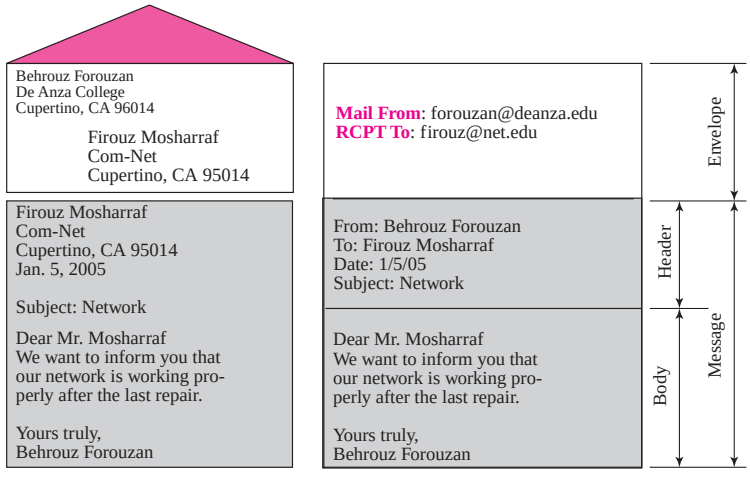
Modern user agents are GUI-based. They contain graphical user interface (GUI) components that allow the user to interact with the software by using both the keyboard and the mouse. They have graphical components such as icons, menu bars, and windows that make the services easy to access.

**Some examples of GUI-based user agents are *Eudora, Outlook,* and *Netscape*.**

## Sending Mail

To send mail, the user, through the UA, creates mail that looks very similar to postal mail. It has an *envelope* and a *message* (see Figure 23.6).

**Figure 23.6** *Format of an e-mail*




### *Envelope*

The **envelope** usually contains the sender address, the receiver address, and other information.

### *Message*

The message contains the **header** and the **body.** The header of the message defines the sender, the receiver, the subject of the message, and some other information. The body of the message contains the actual information to be read by the recipient.

## Receiving Mail

The user agent is triggered by the user (or a timer). If a user has mail, the UA informs the user with a notice. If the user is ready to read the mail, a list is displayed in which each line contains a summary of the information about a particular message in the mailbox. The summary usually includes the sender mail address, the subject, and the time the mail was sent or received. The user can select any of the messages and display its contents on the screen.

## Addresses

To deliver mail, a mail handling system must use an addressing system with unique addresses. In the Internet, the address consists of two parts: a **local part** and a **domain name,** separated by an @ sign (see Figure 23.7).

**Figure 23.7** *E-mail address*


### *Local Part*

The local part defines the name of a special file, called the user mailbox, where all of the mail received for a user is stored for retrieval by the message access agent.

### *Domain Name*

The second part of the address is the domain name. An organization usually selects one or more hosts to receive and send e-mail; they are sometimes called *mail servers* or *exchangers*. The domain name assigned to each mail exchanger either comes from the DNS database or is a logical name (for example, the name of the organization).

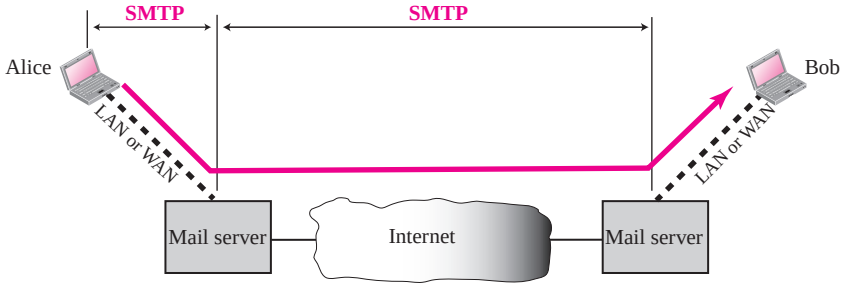
## Mailing List or Group List

Electronic mail allows one name, an **alias,** to represent several different e-mail addresses; this is called a mailing list. Every time a message is to be sent, the system checks the recipient's name against the alias database; if there is a mailing list for the defined alias, separate messages, one for each entry in the list, must be prepared and handed to the MTA. If there is no mailing list for the alias, the name itself is the receiving address and a single message is delivered to the mail transfer entity.

## 23.3 MESSAGE TRANSFER AGENT: SMTP

The actual mail transfer is done through message transfer agents (MTAs). To send mail, a system must have the client MTA, and to receive mail, a system must have a server MTA. The formal protocol that defines the MTA client and server in the Internet is called **Simple Mail Transfer Protocol (SMTP).** As we said before, two pairs of MTA client-server programs are used in the most common situation (fourth scenario). Figure 23.8 shows the range of the SMTP protocol in this scenario.

**Figure 23.8** *SMTP range*




SMTP is used two times, between the sender and the sender's mail server and between the two mail servers. As we will see shortly, another protocol is needed between the mail server and the receiver.

SMTP simply defines how commands and responses must be sent back and forth. Each network is free to choose a software package for implementation. We will discuss the mechanism of mail transfer by SMTP in the remainder of the section.

### Commands and Responses

SMTP uses commands and responses to transfer messages between an MTA client and an MTA server (see Figure 23.9).

**Figure 23.9** *Commands and responses*


Each command or reply is terminated by a two-character (carriage return and line feed) end-of-line token.

### Commands

Commands are sent from the client to the server. The format of a command is shown below:

**Keyword: argument(s)**

It consists of a keyword followed by zero or more arguments. SMTP defines 14 commands listed in Table 23.1 and described in more detail below.

**Table 23.1** *Commands*

| *Keyword* | *Argument(s)* | *Keyword* | *Argument(s)* |
|---|---|---|---|
| HELO | Sender's host name | NOOP | |
| MAIL FROM | Sender of the message | TURN | |
| RCPT TO | Intended recipient | EXPN | Mailing list |
| DATA | Body of the mail | HELP | Command name |
| QUIT | | SEND FROM | Intended recipient |
| RSET | | SMOL FROM | Intended recipient |
| VRFY | Name of recipient | SMAL FROM | Intended recipient |

- **HELO.** This command is used by the client to identify itself. The argument is the domain name of the client host. The format is

**HELO: challenger.atc.fhda.edu**

- **MAIL FROM.** This command is used by the client to identify the sender of the message. The argument is the e-mail address of the sender (local part plus the domain name). The format is

**MAIL FROM: forouzan@challenger.atc.fhda.edu**

- **RCPT TO.** This command is used by the client to identify the intended recipient of the message. The argument is the e-mail address of the recipient. If there are multiple recipients, the command is repeated. The format is

**RCPT TO: betsy@mcgraw-hill.com**

- **DATA.** This command is used to send the actual message. All lines that follow the DATA command are treated as the mail message. The message is terminated by a line containing just one period. The format is

```
DATA

This is the message
to be sent to the McGraw-Hill
Company.
.
```

- **QUIT.** This command terminates the message. The format is

```
QUIT
```

- **RSET.** This command aborts the current mail transaction. The stored information about the sender and recipient is deleted. The connection will be reset.

```
RSET
```

- **VRFY.** This command is used to verify the address of the recipient, which is sent as the argument. The sender can ask the receiver to confirm that a name identifies a valid recipient. Its format is

```
VRFY: betsy@mcgraw-hill.com
```

- **NOOP.** This command is used by the client to check the status of the recipient. It requires an answer from the recipient. Its format is

```
NOOP
```

- **TURN.** This command lets the sender and the recipient switch positions, whereby the sender becomes the recipient and vice versa. However, most SMTP implementations today do not support this feature. The format is

```
TURN
```

- **EXPN.** This command asks the receiving host to expand the mailing list sent as the arguments and to return the mailbox addresses of the recipients that comprise the list. The format is

```
EXPN: x y z
```

- **HELP.** This command asks the recipient to send information about the command sent as the argument. The format is

```
HELP: mail
```

- **SEND FROM.** This command specifies that the mail is to be delivered to the terminal of the recipient, and not the mailbox. If the recipient is not logged in, the mail is bounced back. The argument is the address of the sender. The format is

```
SEND FROM: forouzan@fhda.atc.edu
```

- **SMOL FROM.** This command specifies that the mail is to be delivered to the terminal or the mailbox of the recipient. This means that if the recipient is logged in, the

mail is delivered only to the terminal. If the recipient is not logged in, the mail is delivered to the mailbox. The argument is the address of the sender. The format is

**SMOL FROM: forouzan@fhda.atc.edu**

- **SMAL FROM.** This command specifies that the mail is to be delivered to the terminal and the mailbox of the recipient. This means that if the recipient is logged in, the mail is delivered to the terminal and the mailbox. If the recipient is not logged in, the mail is delivered only to the mailbox. The argument is the address of the sender. The format is

**SMAL FROM: forouzan@fhda.atc.edu**

### *Responses*

Responses are sent from the server to the client. A response is a three-digit code that may be followed by additional textual information. The idea is the same as discussed in the case of HTTP responses in Chapter 22. Table 23.2 lists some of the responses.

**Table 23.2** *Responses*

| *Code* | *Description* |
|---|---|
| **Positive Completion Reply** | |
| **211** | System status or help reply |
| **214** | Help message |
| **220** | Service ready |
| **221** | Service closing transmission channel |
| **250** | Request command completed |
| **251** | User not local; the message will be forwarded |
| **Positive Intermediate Reply** | |
| **354** | Start mail input |
| **Transient Negative Completion Reply** | |
| **421** | Service not available |
| **450** | Mailbox not available |
| **451** | Command aborted: local error |
| **452** | Command aborted; insufficient storage |
| **Permanent Negative Completion Reply** | |
| **500** | Syntax error; unrecognized command |
| **501** | Syntax error in parameters or arguments |
| **502** | Command not implemented |
| **503** | Bad sequence of commands |
| **504** | Command temporarily not implemented |
| **550** | Command is not executed; mailbox unavailable |
| **551** | User not local |
| **552** | Requested action aborted; exceeded storage location |
| **553** | Requested action not taken; mailbox name not allowed |
| **554** | Transaction failed |

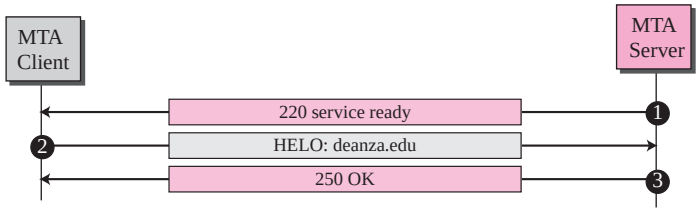
## Mail Transfer Phases

The process of transferring a mail message occurs in three phases: **connection establishment,** mail transfer, and **connection termination.**

### *Connection Establishment*

After a client has made a TCP connection to the well-known port 25, the SMTP server starts the connection phase. This phase involves the following three steps, which are illustrated in Figure 23.10.

**Figure 23.10** *Connection establishment*



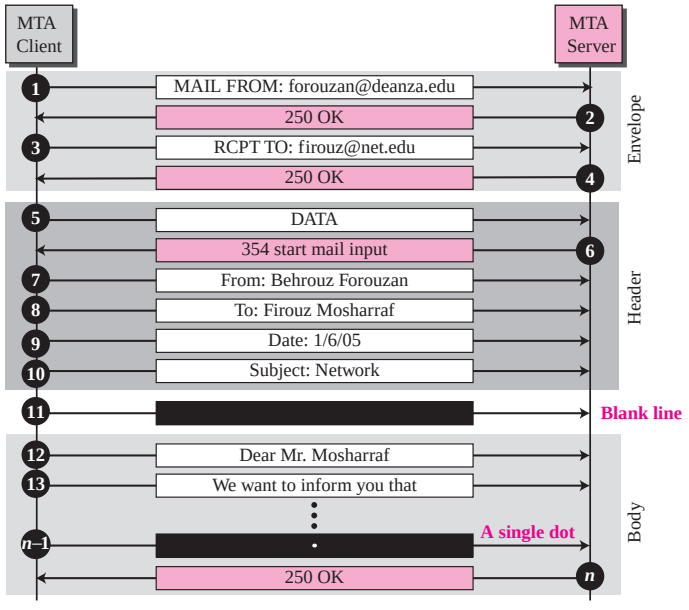

1. The server sends code 220 (service ready) to tell the client that it is ready to receive mail. If the server is not ready, it sends code 421 (service not available).
2. The client sends the HELO message to identify itself using its domain name address. This step is necessary to inform the server of the domain name of the client. Remember that during TCP connection establishment, the sender and receiver know each other through their IP addresses.
3. The server responds with code 250 (request command completed) or some other code depending on the situation.

### *Message Transfer*

After connection has been established between the SMTP client and server, a single message between a sender and one or more recipients can be exchanged. This phase involves eight steps. Steps 3 and 4 are repeated if there is more than one recipient (see Figure 23.11).

1. The client sends the MAIL FROM message to introduce the sender of the message. It includes the mail address of the sender (mailbox and the domain name). This step is needed to give the server the return mail address for returning errors and reporting messages.
2. The server responds with code 250 or some other appropriate code.
3. The client sends the RCPT TO (recipient) message, which includes the mail address of the recipient.
4. The server responds with code 250 or some other appropriate code.
5. The client sends the DATA message to initialize the message transfer.

**Figure 23.11** *Message transfer*

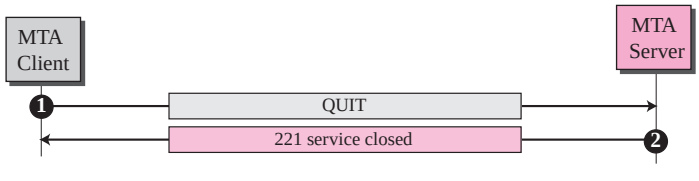



6. The server responds with code 354 (start mail input) or some other appropriate message.
7. The client sends the contents of the message in consecutive lines. Each line is terminated by a two-character end-of-line token (carriage return and line feed). The message is terminated by a line containing just one period.
8. The server responds with code 250 (OK) or some other appropriate code.

### *Connection Termination*

After the message is transferred successfully, the client terminates the connection. This phase involves two steps (see Figure 23.12).

**Figure 23.12** *Connection termination*



1. The client sends the QUIT command.
2. The server responds with code 221 or some other appropriate code.

After the connection termination phase, the TCP connection must be closed.

### Example 23.1

Let us see how we can directly use SMTP to send an e-mail and simulate the commands and responses we described in this section. We use TELNET to log into port 25 (the well-known port for SMTP). We then use the commands directly to send an e-mail. In this example, forouzanb@adelphia.net is sending an e-mail to himself. The first few lines show TELNET trying to connect to the *adelphia* mail server.

```
$ telnet mail.adelphia.net 25
Trying 68.168.78.100...
Connected to mail.adelphia.net (68.168.78.100).
```

After connection, we can type the SMTP commands and then receive the responses as shown below. We have shown the commands in black and the responses in color. Note that we have added for clarification some comment lines, designated by the "=" sign. These lines are not part of the e-mail procedure.

```
================== Connection Establishment ================
  220 mta13.adelphia.net SMTP server ready Fri, 6 Aug 2004 . . .
HELO mail.adelphia.net
  250 mta13.adelphia.net
=====================       Envelope          ==================
MAIL FROM: forouzanb@adelphia.net
  250 Sender <forouzanb@adelphia.net> Ok
RCPT TO: forouzanb@adelphia.net
  250 Recipient <forouzanb@adelphia.net> Ok
====================      Header and Body      =================
DATA
  354 Ok Send data ending with <CRLF>.<CRLF>
From: Forouzan
TO: Forouzan

This is a test message
to show SMTP in action.
.
```

```
====================  Connection Termination ==================
  250 Message received: adelphia.net@mail.adelphia.net
QUIT
     221 mta13.adelphia.net SMTP server closing connection
Connection closed by foreign host.
```

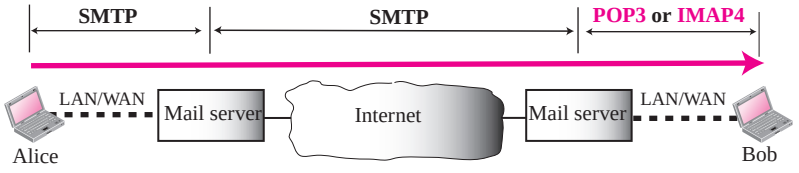
## 23.4 MESSAGE ACCESS AGENT: POP AND IMAP

The first and the second stages of mail delivery use SMTP. However, SMTP is not involved in the third stage because SMTP is a *push* protocol; it pushes the message from the client to the server. In other words, the direction of the bulk data (messages) is

from the client to the server. On the other hand, the third stage needs a *pull* protocol; the client must pull messages from the server. The direction of the bulk data are from the server to the client. The third stage uses a message access agent.

Currently two message access protocols are available: Post Office Protocol, version 3 (POP3) and Internet Mail Access Protocol, version 4 (IMAP4). Figure 23.13 shows the position of these two protocols in the most common situation (fourth scenario).

**Figure 23.13** *POP3 and IMAP4*

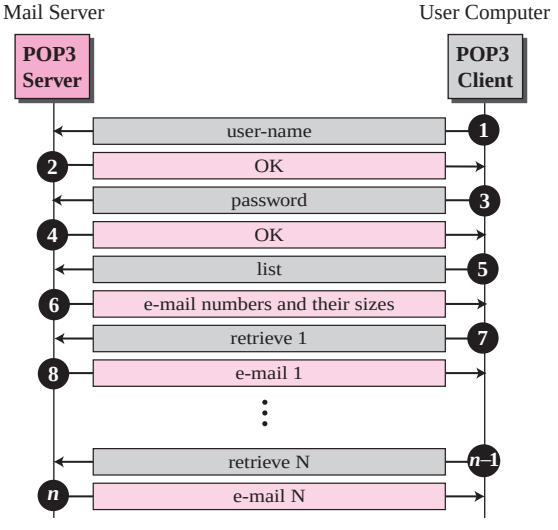



## POP3

**Post Office Protocol, version 3 (POP3)** is simple and limited in functionality. The client POP3 software is installed on the recipient computer; the server POP3 software is installed on the mail server.

Mail access starts with the client when the user needs to download its e-mail from the mailbox on the mail server. The client opens a connection to the server on TCP port 110. It then sends its user name and password to access the mailbox. The user can then list and retrieve the mail messages, one by one. Figure 23.14 shows an example of downloading using POP3.

**Figure 23.14** *POP3*



POP3 has two modes: the delete mode and the keep mode. In the delete mode, the mail is deleted from the mailbox after each retrieval. In the keep mode, the mail remains in the mailbox after retrieval. The delete mode is normally used when the user is working at her permanent computer and can save and organize the received mail after reading or replying. The keep mode is normally used when the user accesses her mail away from her primary computer (e.g., a laptop). The mail is read but kept in the system for later retrieval and organizing.

### IMAP4

Another mail access protocol is **Internet Mail Access Protocol, version 4 (IMAP4).** IMAP4 is similar to POP3, but it has more features; IMAP4 is more powerful and more complex.

POP3 is deficient in several ways. It does not allow the user to organize her mail on the server; the user cannot have different folders on the server. (Of course, the user can create folders on her own computer.) In addition, POP3 does not allow the user to partially check the contents of the mail before downloading.

IMAP4 provides the following extra functions:

- A user can check the e-mail header prior to downloading.
- A user can search the contents of the e-mail for a specific string of characters prior to downloading.
- A user can partially download e-mail. This is especially useful if bandwidth is limited and the e-mail contains multimedia with high bandwidth requirements.
- A user can create, delete, or rename mailboxes on the mail server.
- A user can create a hierarchy of mailboxes in a folder for e-mail storage.

## 23.5 MIME

Electronic mail has a simple structure. Its simplicity, however, comes with a price. It can send messages only in NVT 7-bit ASCII format. In other words, it has some limitations. It cannot be used for languages other than English (such as French, German, Hebrew, Russian, Chinese, and Japanese). Also, it cannot be used to send binary files or video or audio data.

**Multipurpose Internet Mail Extensions (MIME)** is a supplementary protocol that allows non-ASCII data to be sent through e-mail. MIME transforms non-ASCII data at the sender site to NVT ASCII data and delivers it to the client MTA to be sent through the Internet. The message at the receiving site is transformed back to the original data.

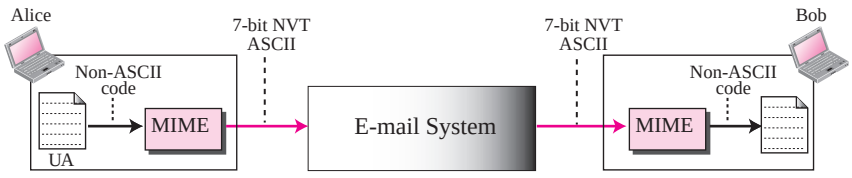
We can think of MIME as a set of software functions that transforms non-ASCII data to ASCII data and vice versa, as shown in Figure 23.15.

### MIME Headers

MIME defines five headers that can be added to the original e-mail header section to define the transformation parameters:

1. MIME-Version
2. Content-Type

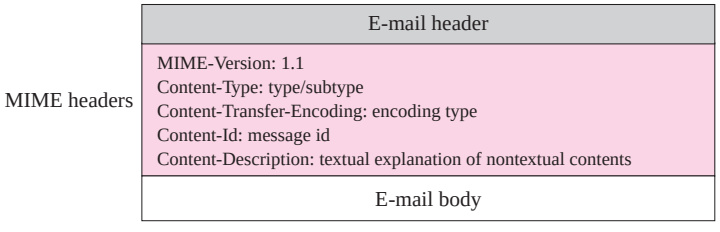
**Figure 23.15** *MIME*




3. Content-Transfer-Encoding
4. Content-Id
5. Content-Description

Figure 23.16 shows the MIME headers. We will describe each header in detail.

**Figure 23.16** *MIME header*




### *MIME-Version*

This header defines the version of MIME used. The current version is 1.1.

### *Content-Type*

This header defines the type of data used in the body of the message. The content type and the content subtype are separated by a slash. Depending on the subtype, the header may contain other parameters. MIME allows seven different types of data, listed in Table 23.3.

- **Text.** The original message is in 7-bit ASCII format and no transformation by MIME is needed. There are two subtypes currently used, *plain* and *HTML*.
- **Multipart.** The body contains multiple, independent parts. The multipart header needs to define the boundary between each part. A parameter is used for this purpose. The parameter is a string token that comes before each part; it is on a separate line by itself and is preceded by two hyphens. The body is terminated using the boundary token, again preceded by two hyphens and then terminated with two hyphens. Four subtypes are defined for this type: *mixed*, *parallel*, *digest*, and *alternative*. In the mixed subtype, the parts must be presented to the recipient in the exact order as in the message. Each part has a different type and is defined at the boundary. The parallel subtype is similar to the mixed subtype, except that the order of the parts is

**Table 23.3** *Data Types and Subtypes in MIME*

| *Type* | *Subtype* | *Description* |
|---|---|---|
| Text | Plain | Unformatted |
| | HTML | HTML format (see Appendix E) |
| Multipart | Mixed | Body contains ordered parts of different data types |
| | Parallel | Same as above, but no order |
| | Digest | Similar to Mixed, but the default is message/RFC822 |
| | Alternative | Parts are different versions of the same message |
| Message | RFC822 | Body is an encapsulated message |
| | Partial | Body is a fragment of a bigger message |
| | External-Body | Body is a reference to another message |
| Image | JPEG | Image is in JPEG format |
| | GIF | Image is in GIF format |
| Video | MPEG | Video is in MPEG format |
| Audio | Basic | Single channel encoding of voice at 8 KHz |
| Application | PostScript | Adobe PostScript |
| | Octet-stream | General binary data (eight-bit bytes) |

unimportant. The digest subtype is also similar to the mixed subtype except that the default type/subtype is message/RFC822 as defined below. In the alternative subtype, the same message is repeated using different formats. The following is an example of a multipart message using a mixed subtype:

```
Content-Type: multipart/mixed; boundary=xxxx

      --xxxx
      Content-Type: text/plain;
      ...
      --xxxx
      Content-Type: image/gif;
      ...
      --xxxx--
```

- **Message.** In the message type, the body is itself an entire mail message, a part of a mail message, or a pointer to a message. Three subtypes are currently used: *RFC822, partial,* and *external-body.* The subtype RFC822 is used if the body is encapsulating another message (including header and the body). The partial subtype is used if the original message has been fragmented into different mail messages and this mail message is one of the fragments. The fragments must be reassembled at the destination by MIME. Three parameters must be added: *id, number,* and the *total*. The id identifies the message and is present in all the fragments. The number defines the sequence order of the fragment. The total defines the number of fragments that comprise the original message. The following is an example of a message with three fragments:

```
Content-Type: message/partial;
id="forouzan@challenger.atc.fhda.edu";
number=1;
total=3;
...
```

The subtype external-body indicates that the body does not contain the actual message but is only a reference (pointer) to the original message. The parameters following the subtype define how to access the original message. The following is an example:

```
Content-Type: message/external-body;
name="report.txt";
site="fhda.edu";
access-type="ftp";
...
```

- **Image.** The original message is a stationary image, indicating that there is no animation. The two currently used subtypes are *Joint Photographic Experts Group (JPEG)*, which uses image compression, and *Graphics Interchange Format (GIF)*.
- **Video.** The original message is a time-varying image (animation). The only subtype is *Moving Picture Experts Group* (*MPEG*). If the animated image contains sounds, it must be sent separately using the audio content type.
- **Audio.** The original message is sound. The only subtype is basic, which uses 8-kHz standard audio data.
- **Application.** The original message is a type of data not previously defined. There are only two subtypes used currently: *PostScript* and *octet-stream.* PostScript is used when the data are in Adobe PostScript format. Octet-stream is used when the data must be interpreted as a sequence of 8-bit bytes (binary file).

### *Content-Transfer-Encoding*

This header defines the method used to encode the messages into 0s and 1s for transport:

```
Content-Transfer-Encoding: <type>
```

The five types of encoding methods are listed in Table 23.4.

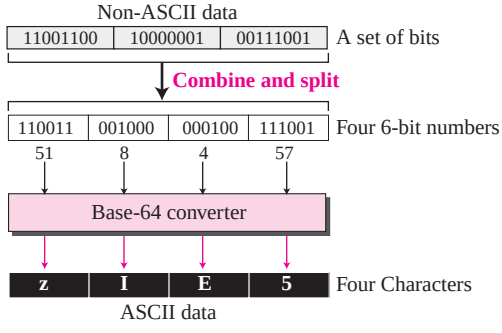
**Table 23.4** *Content-Transfer-Encoding*

| *Type* | *Description* |
|---|---|
| 7bit | NVT ASCII characters and short lines |
| 8bit | Non-ASCII characters and short lines |
| Binary | Non-ASCII characters with unlimited-length lines |
| Base64 | 6-bit blocks of data are encoded into 8-bit ASCII characters |
| Quoted-printable | Non-ASCII characters are encoded as an equal sign plus an ASCII code |

- **7bit.** This is 7-bit NVT ASCII encoding. Although no special transformation is needed, the length of the line should not exceed 1,000 characters.

- **8bit.** This is 8-bit encoding. Non-ASCII characters can be sent, but the length of the line still should not exceed 1,000 characters. MIME does not do any encoding here; the underlying SMTP protocol must be able to transfer 8-bit non-ASCII characters. It is, therefore, not recommended. Base64 and quoted-printable types are preferable.
- **Binary.** This is 8-bit encoding. Non-ASCII characters can be sent, and the length of the line can exceed 1,000 characters. MIME does not do any encoding here; the underlying SMTP protocol must be able to transfer binary data. It is, therefore, not recommended. Base64 and quoted-printable types are preferable.
- **Base64.** This is a solution for sending data made of bytes when the highest bit is not necessarily zero. Base64 transforms this type of data to printable characters, which can then be sent as ASCII characters or any type of character set supported by the underlying mail transfer mechanism. Base64 divides the binary data (made of streams of bits) into 24-bit blocks. Each block is then divided into four sections, each made of 6 bits (see Figure 23.17).

**Figure 23.17** *Base64*



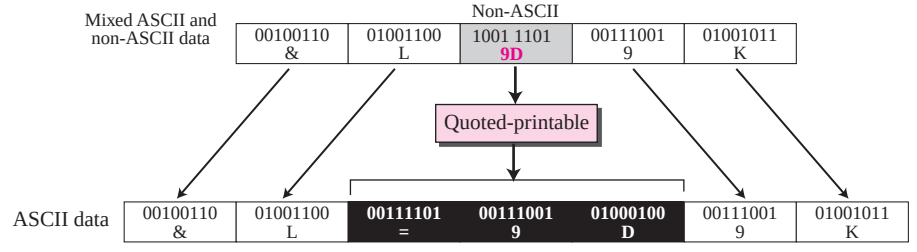

Each 6-bit section is interpreted as one character according to Table 23.5.

**Table 23.5** *Base-64 Converting Table*

| *Value* | *Code* | *Value* | *Code* | *Value* | *Code* | *Value* | *Code* | *Value* | *Code* | *Value* | *Code* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | **A** | 11 | **L** | 22 | **W** | 33 | **h** | 44 | **s** | 55 | **3** |
| 1 | **B** | 12 | **M** | 23 | **X** | 34 | **i** | 45 | **t** | 56 | **4** |
| 2 | **C** | 13 | **N** | 24 | **Y** | 35 | **j** | 46 | **u** | 57 | **5** |
| 3 | **D** | 14 | **O** | 25 | **Z** | 36 | **k** | 47 | **v** | 58 | **6** |
| 4 | **E** | 15 | **P** | 26 | **a** | 37 | **l** | 48 | **w** | 59 | **7** |
| 5 | **F** | 16 | **Q** | 27 | **b** | 38 | **m** | 49 | **x** | 60 | **8** |
| 6 | **G** | 17 | **R** | 28 | **c** | 39 | **n** | 50 | **y** | 61 | **9** |
| 7 | **H** | 18 | **S** | 29 | **d** | 40 | **o** | 51 | **z** | 62 | + |
| 8 | **I** | 19 | **T** | 30 | **e** | 41 | **p** | 52 | **0** | 63 | / |
| 9 | **J** | 20 | **U** | 31 | **f** | 42 | **q** | 53 | **1** | | |
| 10 | **K** | 21 | **V** | 32 | **g** | 43 | **r** | 54 | **2** | | |

- **Quoted-printable.** Base64 is a redundant encoding scheme; that is, 24 bits become four characters, and eventually are sent as 32 bits. We have an overhead of 25 percent. If the data consist mostly of ASCII characters with a small non-ASCII portion, we can use quoted-printable encoding. If a character is ASCII, it is sent as is. If a character is not ASCII, it is sent as three characters. The first character is the equal sign (=). The next two characters are the hexadecimal representations of the byte. Figure 23.18 shows an example.

**Figure 23.18** *Quoted-printable*




### *Content-Id*

This header uniquely identifies the whole message in a multiple message environment.

### *Content-Description*

This header defines whether the body is image, audio, or video.

## 23.6 WEB-BASED MAIL

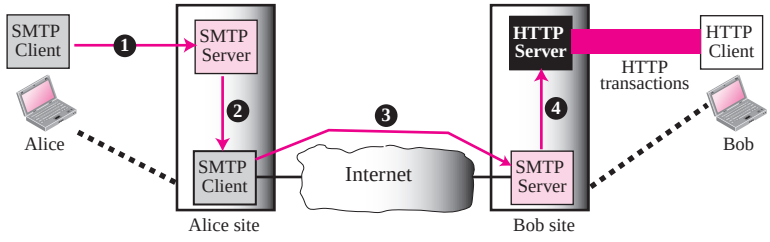
E-mail is such a common application that some websites today provide this service to anyone who accesses the site. Three common sites are Hotmail, Yahoo, and Google. The idea is very simple. Let us go through two cases:

### Case I

In the first case, Alice, the sender, uses a traditional mail server; Bob, the receiver, has an account on a Web-based server. Mail transfer from Alice's browser to her mail server is done through SMTP. The transfer of the message from the sending mail server to the receiving mail server is still through SMTP. However, the message from the receiving server (the web server) to Bob's browser is done through HTTP. In other words, instead of using POP3 or IMAP4, HTTP is normally used. When Bob needs to retrieve his e-mails, he sends a request HTTP message to the website (Hotmail, for example). The website sends a form to be filled in by Bob, which includes the log-in name and the password. If the log-in name and password match, the list of e-mails is transferred from the Web server to Bob's browser in HTML format. Now Bob can

browse through his received e-mails and then, using more HTTP transactions, can get his e-mails one by one. This is shown in Figure 23.19.

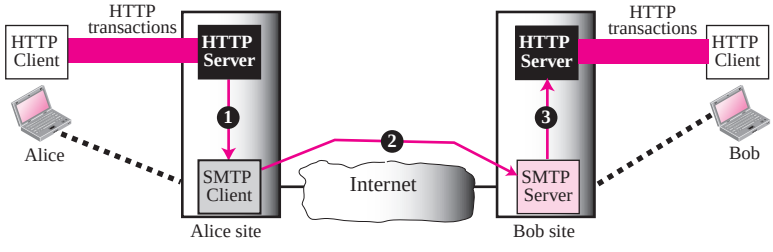
**Figure 23.19** *Web-base e-mail, case I*




## Case II

In the second case, both Alice and Bob use Web servers, but not necessarily the same server. Alice sends the message to the Web server using HTTP transactions. Alice sends an HTTP request message to her Web server using the name and address of Bob's mailbox as the URL. The server at the Alice site passes the message to the SMTP client and sends it to the server at the Bob site using SMTP protocol. Bob receives the message using HTTP transactions. However, the message from the server at the Alice site to the server at the Bob site still takes place using SMTP protocol. Figure 23.20 shows the idea.

**Figure 23.20** *Web-based e-mail, case II*




# 23.7 E-MAIL SECURITY

The protocol discussed in this chapter does not provide any security provisions per se. However, e-mail exchanges can be secured using two application-layer securities designed in particular for e-mail systems. Two of these protocols, Pretty Good Privacy (PGP) and Secure MIME (SMIME) are discussed in Chapter 30 after we have discussed the basic network security.

## 23.8 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give an easy but thorough coverage of electronic mail including [Com 06], [Ste 94], and [Tan 03], and [Kur & Ros 08].

### RFCs

Several RFCs show updates on SMTP, including RFC 2821 and RFC 2822. POP3 is explained in RFC 1939. Several RFCs refer to MIME, including RFC 2046, RFC 2047, RFC 2048, and RFC 2049.

## 23.9 KEY TERMS

alias
body
connection establishment
connection termination
domain name
envelope
header
Internet Mail Access Protocol, version 4 (IMAP4)
local part
message access agent (MAA)
message transfer agent (MTA)
Multipurpose Internet Mail Extensions (MIME)
Post Office Protocol, version 3 (POP3)
Simple Mail Transfer Protocol (SMTP)
user agent (UA)

## 23.10 SUMMARY

- Electronic mail is one of the most common applications on the Internet. The e-mail architectures consists of several components such as user agent (UA), main transfer agent (MTA), and main access agent (MAA).
- The UA prepares the message, creates the envelope, and puts the message in the envelope. The mail address consists of two parts: a local part (user mailbox) and a domain name. The form is localpart@domainname. An alias allows the use of a mailing list.
- The MTA transfers the mail across the Internet, a LAN, or a WAN. The protocol that implements MTA is called Simple Main Transfer Protocol (SMTP). SMTP uses commands and responses to transfer messages between an MTA client and an MTA server. The steps in transferring a mail message are: connection establishment mail transfer, and connection termination.

- Two protocols are used to implement MAA: Post Office Protocol, version 3 (POP3) and Internet Mail Access Protocol, version 4 (IMAP4). These protocols are used by the receiver to pull messages from a mail server.
- Multipurpose Internet Mail Extension (MIME) allows the transfer of multimedia messages. MIME changes multimedia characters into ASCII characters that are transferable through the e-mail system.
- Web-based e-mails get popularity through sites that offer free e-mails for the users. In Web-based e-mail systems part of the data transfer is done through the SMTP protocol and part through the HTTP protocol.
- Secure e-mail is possible through two technologies: Pretty Good Privacy (PGP) and SMIME (Secure MIME).

## 23.11 PRACTICE SET

### Exercises

1. A sender sends unformatted text. Show the MIME header.
2. A sender sends a JPEG message. Show the MIME header.
3. A non-ASCII message of 1,000 bytes is encoded using base64. How many bytes are in the encoded message? How many bytes are redundant? What is the ratio of redundant bytes to the total message?
4. A message of 1,000 bytes is encoded using quoted-printable. The message consists of 90 percent ASCII and 10 percent non-ASCII characters. How many bytes are in the encoded message? How many bytes are redundant? What is the ratio of redundant bytes to the total message?
5. Compare the results of Exercises 3 and 4. How much is the efficiency improved if the message is a combination of ASCII and non-ASCII characters?
6. Encode the following message in base64:

```
01010111 00001111 11110000 10101111 01110001 01010100
```

7. Encode the following message in quoted-printable:

```
01010111 00001111 11110000 10101111 01110001 01010100
```

8. Encode the following message in base64:

```
01010111 00001111 11110000 10101111 01110001
```

9. Encode the following message in quoted-printable:

```
01010111 00001111 11110000 10101111 01110001
```

10. Are the HELO and MAIL FROM commands both necessary? Why or why not?
11. In Figure 23.11 what is the difference between MAIL FROM in the envelope and the From in the header?
12. Why is a connection establishment for mail transfer needed if TCP has already established a connection?
13. Show the connection establishment phase from aaa@xxx.com to bbb@yyy.com.
14. Show the message transfer phase from aaa@xxx.com to bbb@yyy.com. The message is "Good morning my friend."
15. Show the connection termination phase from aaa@xxx.com to bbb@yyy.com.
16. User aaa@xxx.com sends a message to user bbb@yyy.com, which is forwarded to user ccc@zzz.com. Show all SMTP commands and responses.
17. User aaa@xxx.com sends a message to user bbb@yyy.com. The latter replies. Show all SMTP commands and responses.
18. In SMTP, if we send a one-line message between two users, how many lines of commands and responses are exchanged?

## Research Activities

19. A new version of SMTP, called ESMTP, is in use today. Find the differences between the two.
20. Find information about the *smileys* used to express a user's emotions.

CHAPTER

# 24

# *Network Management: SNMP*

The **Simple Network Management Protocol (SNMP)** is a framework for managing devices in an internet using the TCP/IP protocol suite. It provides a set of fundamental operations for monitoring and maintaining an internet.

## OBJECTIVES

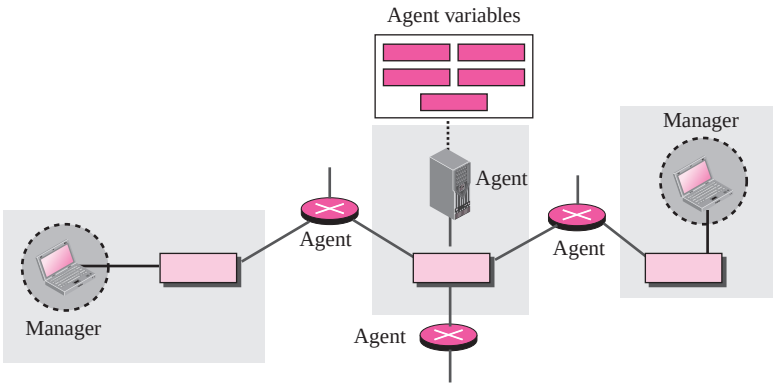
*The chapter has several objectives:*

- To discuss SNMP as a framework for managing devices in an internet using the TCP/IP protocol suite.
- To define a manager as a host that runs SNMP client and any agents as a router or host that runs a server program.
- Discuss SMI and MIB, which are used by SNMP.
- To show how SMI names objects, defines the type of data, and encodes data.
- To show how data types are defined using ASN.1.
- To show how SMI uses BER to encode data.
- To show the functionality of SNMP using three methods.
- To discuss the format of SNMP messages.
- To show how SNMP uses two different ports of UDP.
- To show how SNMPv3 has enhanced security features over previous versions.

## 24.1 CONCEPT

SNMP uses the concept of manager and agent. That is, a manager, usually a host, controls and monitors a set of agents, usually routers or servers (see Figure 24.1).

**Figure 24.1** *SNMP concept*




SNMP is an application-level protocol in which a few manager stations control a set of agents. The protocol is designed at the application level so that it can monitor devices made by different manufacturers and installed on different physical networks. In other words, SNMP frees management tasks from both the physical characteristics of the managed devices and the underlying networking technology. It can be used in a heterogeneous internet made of different LANs and WANs connected by routers made by different manufacturers.

### Managers and Agents

A management station, called a **manager,** is a host that runs the SNMP client program. A managed station, called an **agent,** is a router (or a host) that runs the SNMP server program. Management is achieved through simple interaction between a manager and an agent.

The agent keeps performance information in a database. The manager has access to the values in the database. For example, a router can store in appropriate variables the number of packets received and forwarded. The manager can fetch and compare the values of these two variables to see if the router is congested or not.

The manager can also make the router perform certain actions. For example, a router periodically checks the value of a reboot counter to see when it should reboot

itself. It reboots itself, for example, if the value of the counter is 0. The manager can use this feature to reboot the agent remotely at any time. It simply sends a packet to force a 0 value in the counter.

Agents can also contribute to the management process. The server program running on the agent can check the environment and, if it notices something unusual, it can send a warning message (called a **trap**) to the manager.

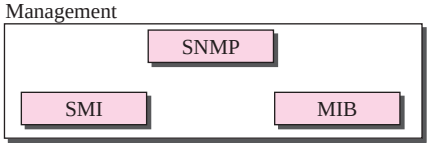
In other words, management with SNMP is based on three basic ideas:

1. A manager checks an agent by requesting information that reflects the behavior of the agent.
2. A manager forces an agent to perform a task by resetting values in the agent database.
3. An agent contributes to the management process by warning the manager of an unusual situation.

## 24.2 MANAGEMENT COMPONENTS

To do management tasks, SNMP uses two other protocols: **Structure of Management Information (SMI)** and **Management Information Base (MIB).** In other words, management on the Internet is done through the cooperation of three protocols: SNMP, SMI, and MIB, as shown in Figure 24.2.

**Figure 24.2** *Components of network management on the Internet*




Let us elaborate on the interactions between these protocols.

### Role of SNMP

SNMP has some very specific roles in network management. It defines the format of the packet to be sent from a manager to an agent and vice versa. It also interprets the result and creates statistics (often with the help of other management software). The packets exchanged contain the object (variable) names and their status (values). SNMP is responsible for reading and changing these values.

**SNMP defines the format of packets exchanged between a manager and an agent. It reads and changes the status of objects (values of variables) in SNMP packets.**

### Role of SMI

To use SNMP, we need rules. We need rules for naming objects. This is particularly important because the objects in SNMP form a hierarchical structure (an object may

have a parent object and some child objects). Part of a name can be inherited from the parent. We also need rules to define the type of the objects. What types of objects are handled by SNMP? Can SNMP handle simple types or structured types? How many simple types are available? What are the sizes of these types? What is the range of these types? In addition, how are each of these types encoded?

**SMI defines the general rules for naming objects, defining object types (including range and length), and showing how to encode objects and values.**

We need these universal rules because we do not know the architecture of the computers that send, receive, or store these values. The sender may be a powerful computer in which an integer is stored as 8-byte data; the receiver may be a small computer that stores an integer as 4-byte data.

SMI is a protocol that defines these rules. However, we must understand that SMI only defines the rules; it does not define how many objects are managed in an entity or which object uses which type. SMI is a collection of general rules to name objects and to list their types. The association of an object with the type is not done by SMI.

## Role of MIB

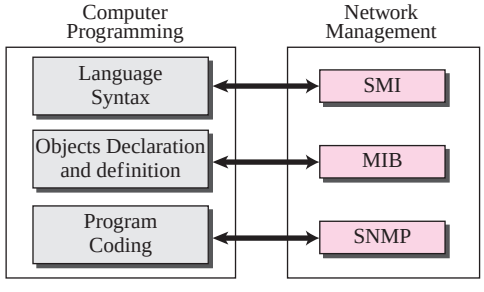
We hope it is clear that we need another protocol. For each entity to be managed, this protocol must define the number of objects, name them according to the rules defined by SMI, and associate a type to each named object. This protocol is MIB. MIB creates a set of objects defined for each entity similar to a database (mostly meta data in a database, names and types without values).

**MIB creates a collection of named objects, their types, and their relationships to each other in an entity to be managed.**

## An Analogy

Before discussing each of these protocols in more detail, let us give an analogy. The three network management components are similar to what we need when we write a program in a computer language to solve a problem. Figure 24.3 shows the analogy.

**Figure 24.3** *Comparing computer programming and network management*



### *Syntax: SMI*

Before we write a program, the syntax of the language (such as C or Java) must be predefined. The language also defines the structure of variables (simple, structured, pointer, and so on) and how the variables must be named. For example, a variable name must be 1 to *n* characters in length and start with a letter followed by alphanumeric characters. The language also defines the type of data to be used (integer, float, char, etc.). In programming the rules are defined by the syntax of the language. In network management the rules are defined by SMI.

### *Object Declaration and Definition: MIB*

Most computer languages require that objects be declared and defined in each specific program. Declaration and definition creates objects using predefined type and allocates memory location for them. For example, if a program has two variables (an integer named *counter* and an array named *grades* of type char), they must be declared at the beginning of the program:

```
int counter;
char grades [40];
```

MIB does this task in network management. MIB names each object and defines the type of the objects. Because the type is defined by SMI, SNMP knows the range and size.

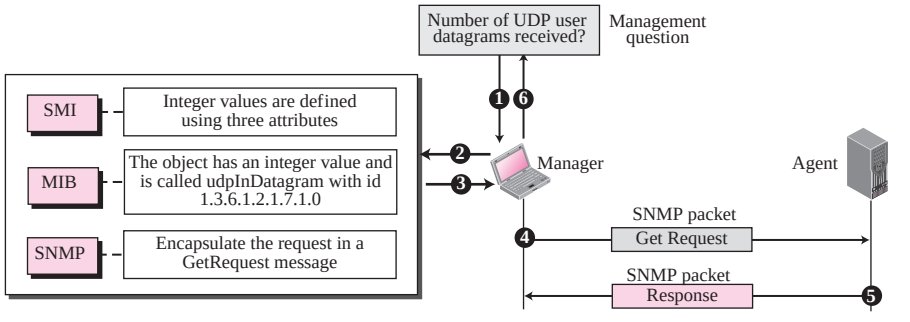
### *Program Coding: SNMP*

After declaration in programming, the program needs to write statements to store values in the variables and change them if needed. SNMP does this task in network management. SNMP stores, changes, and interprets the values of objects already declared by MIB according to the rules defined by SMI.

## An Overview

Before discussing each component in detail, let us show how each of these components is involved in a simple scenario. This is an overview that will be developed later at the end of the chapter. A manager station (SNMP client) wants to send a message to an agent station (SNMP server) to find the number of UDP user datagrams received by the agent. Figure 24.4 shows an overview of steps involved.

**Figure 24.4** *Management overview*



MIB is responsible for finding the object that holds the number of UDP user datagrams received. SMI, with the help of another embedded protocol, is responsible for encoding the name of the object. SNMP is responsible for creating a message, called a GetRequest message, and encapsulating the encoded message. Of course, things are more complicated than this simple overview, but we first need more details of each protocol.

## 24.3 SMI

The Structure of Management Information, version 2 (SMIv2) is a component for network management. Its functions are:

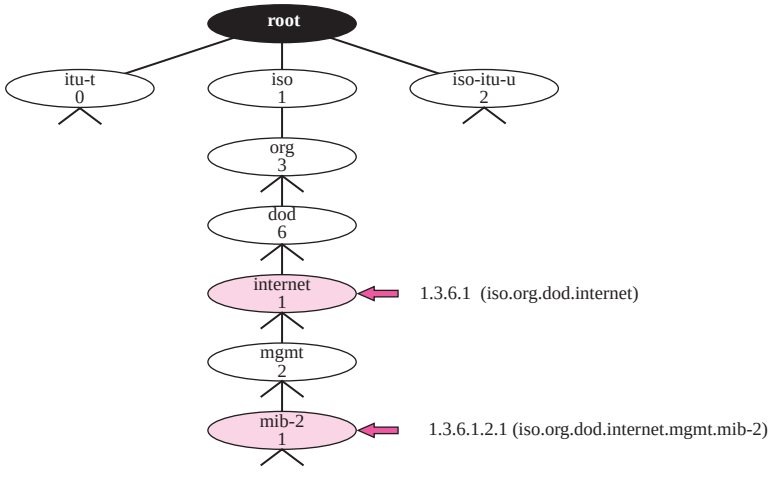
1. To name objects.
2. To define the type of data that can be stored in an object.
3. To show how to encode data for transmission over the network.

SMI is a guideline for SNMP. It emphasizes three attributes to handle an object: name, data type, and encoding method.

### Name

SMI requires that each managed object (such as a router, a variable in a router, a value, etc.) have a unique name. To name objects globally, SMI uses an **object identifier,** which is a hierarchical identifier based on a tree structure (see Figure 24.5).

**Figure 24.5** *Object identifier*




The tree structure starts with an unnamed root. Each object can be defined using a sequence of integers separated by dots. The tree structure can also define an object using a sequence of textual names separated by dots. The integer-dot representation is used in SNMP. The name-dot notation is used by people. For example, the following shows the same object in two different notations:

| **iso.org.dod.internet.mgmt.mib-2** | ↔ | **1.3.6.1.2.1** |
|---|---|---|

The objects that are used in SNMP are located under the *mib-2* object, so their identifiers always start with 1.3.6.1.2.1.

**All objects managed by SNMP are given an object identifier.**
**The object identifier always starts with 1.3.6.1.2.1.**

## Type

The second attribute of an object is the type of data stored in it. To define the data type, SMI uses fundamental **Abstract Syntax Notation 1 (ASN.1)** definitions and adds some new definitions. In other words, SMI is both a subset and a superset of ASN.1.

SMI has two broad categories of data type: *simple* and *structured*. We first define the simple types and then show how the structured types can be constructed from the simple ones.

### *Simple Type*

The **simple data types** are atomic data types. Some of them are taken directly from ASN.1; some are added by SMI. The most important ones are given in Table 24.1. The first five are from ASN.1; the next seven are defined by SMI.

**Table 24.1** *Data Types*

| *Type* | *Size* | *Description* |
|---|---|---|
| INTEGER | 4 bytes | An integer with a value between $-2^{31}$ and $2^{31}-1$ |
| Integer32 | 4 bytes | Same as INTEGER |
| Unsigned32 | 4 bytes | Unsigned with a value between 0 and $2^{32}-1$ |
| OCTET STRING | Variable | Byte-string up to 65,535 bytes long |
| OBJECT IDENTIFIER | Variable | An object identifier |
| IPAddress | 4 bytes | An IP address made of four integers |
| Counter32 | 4 bytes | An integer whose value can be incremented from zero to $2^{32}$; when it reaches its maximum value it wraps back to zero |
| Counter64 | 8 bytes | 64-bit counter |
| Gauge32 | 4 bytes | Same as Counter32, but when it reaches its maximum value, it does not wrap; it remains there until it is reset |
| TimeTicks | 4 bytes | A counting value that records time in 1/100ths of a second |
| BITS | | A string of bits |
| Opaque | Variable | Uninterpreted string |

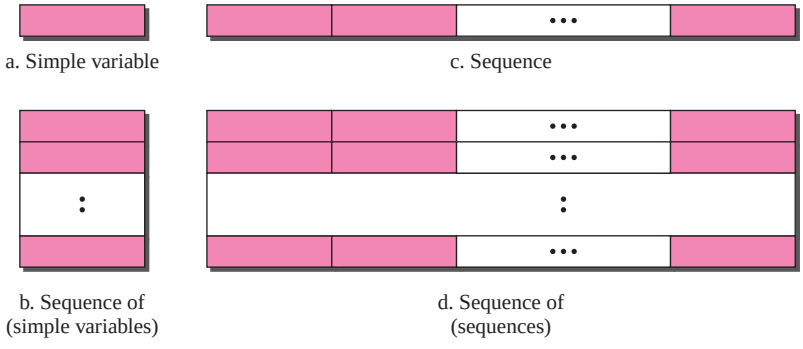
### *Structured Type*

By combining simple and structured data types, we can make new structured data types. SMI defines two **structured data types:** *sequence* and *sequence of*.

- **Sequence.** A *sequence* data type is a combination of simple data types, not necessarily of the same type. It is analogous to the concept of a *struct* or a *record* used in programming languages such as C.
- **Sequence of.** A *sequence of* data type is a combination of simple data types all of the same type or a combination of sequence data types all of the same type. It is analogous to the concept of an *array* used in programming languages such as C.

Figure 24.6 shows a conceptual view of data types.

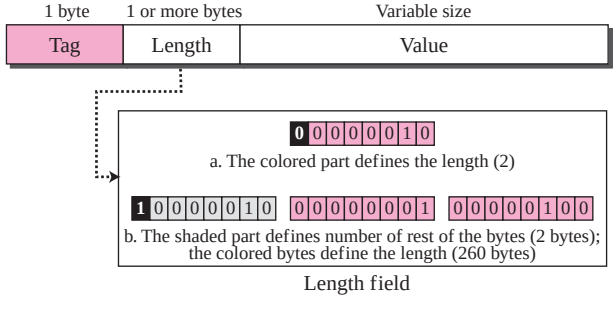
**Figure 24.6** *Conceptual data types*




## Encoding Method

SMI uses another standard, **Basic Encoding Rules (BER),** to encode data to be transmitted over the network. BER specifies that each piece of data be encoded in triplet format: tag, length, and value, as illustrated in Figure 24.7.

**Figure 24.7** *Encoding format*



The tag is a 1-byte field that defines the type of data. Table 24.2 shows the data types we use in this chapter and their tags in binary and hexadecimal numbers. The length field is 1 or more bytes. If it is 1 byte, the most significant bit must be 0. The other 7 bits define the length of the data. If it is more than 1 byte, the most significant bit of the first byte must be 1. The other 7 bits of the first byte define the number of bytes needed to define the length. The value field codes the value of the data according to the rules defined in BER.

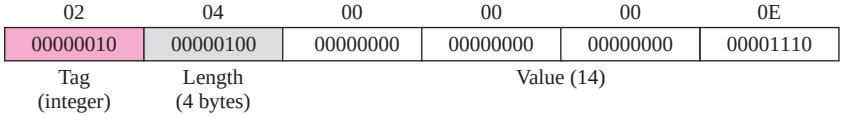
**Table 24.2** *Codes for Data Types*

| *Data Type* | *Tag (Binary)* | *Tag (Hex)* |
|---|---|---|
| INTEGER | **00000010** | **02** |
| OCTET STRING | **00000100** | **04** |
| OBJECT IDENTIFIER | **00000110** | **06** |
| NULL | **00000101** | **05** |
| Sequence, sequence of | **00110000** | **30** |
| IPAddress | **01000000** | **40** |
| Counter | **01000001** | **41** |
| Gauge | **01000010** | **42** |
| TimeTicks | **01000011** | **43** |
| Opaque | **01000100** | **44** |

### Example 24.1

Figure 24.8 shows how to define INTEGER 14. Note that we have used both binary representation and hexadecimal representation for the tag. The size of the length field is from Table 24.1.

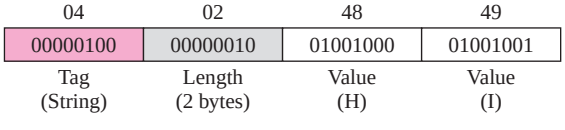
**Figure 24.8** *Example 24.1: INTEGER 14*




### Example 24.2

Figure 24.9 shows how to define the OCTET STRING "HI."

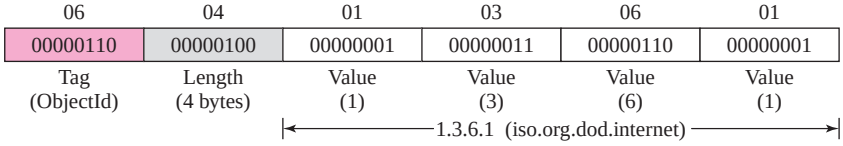
**Figure 24.9** *Example 24.2: OCTET STRING "HI"*



### Example 24.3

Figure 24.10 shows how to define ObjectIdentifier 1.3.6.1 (iso.org.dod.internet).

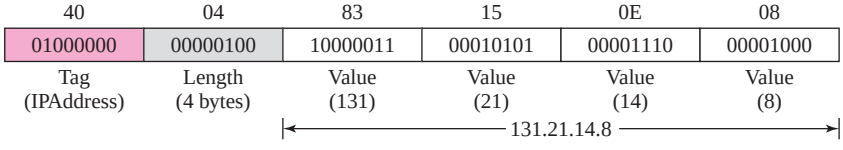
**Figure 24.10** *Example 24.3: ObjectIdentifier 1.3.6.1*




### Example 24.4

Figure 24.11 shows how to define IPAddress 131.21.14.8.

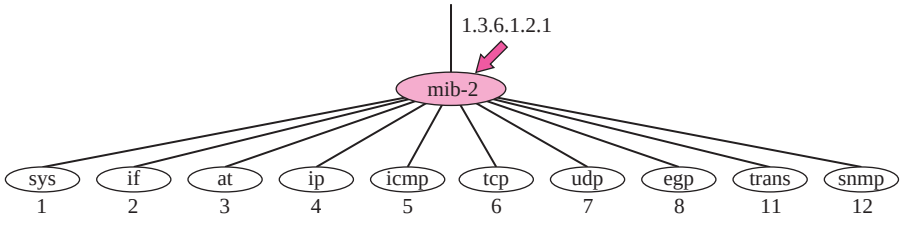
**Figure 24.11** *Example 24.4: IPAddress 131.21.14.8*




## 24.4 MIB

The Management Information Base, version 2 (MIB2) is the second component used in network management. Each agent has its own MIB2, which is a collection of all the objects that the manager can manage. The objects in MIB2 are categorized under 10 different groups: system, interface, address translation, ip, icmp, tcp, udp, egp, transmission, and snmp. These groups are under the mib-2 object in the object identifier tree (see Figure 24.12). Each group has defined variables and/or tables.

**Figure 24.12** *mib-2*




The following is a brief description of some of the objects:

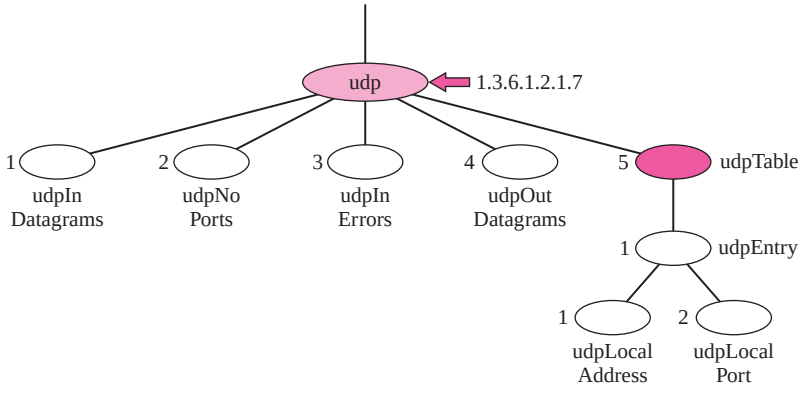
- **sys** This object (*system*) defines general information about the node (system), such as the name, location, and lifetime.

- **if** This object (*interface*) defines information about all of the interfaces of the node including interface number, physical address, and IP address.
- **at** This object (*address translation*) defines the information about the ARP table.
- **ip** This object defines information related to IP, such as the routing table and the IP address.
- **icmp** This object defines information related to ICMP, such as the number of packets sent and received and total errors created.
- **tcp** This object defines general information related to TCP, such as the connection table, time-out value, number of ports, and number of packets sent and received.
- **udp** This object defines general information related to UDP, such as the number of ports and number of packets sent and received.
- **snmp** This object defines general information related to SNMP itself.

## Accessing MIB Variables

To show how to access different variables, we use the udp group as an example. There are four simple variables in the udp group and one sequence of (table of) records. Figure 24.13 shows the variables and the table.

**Figure 24.13** *udp group*




We will show how to access each entity.

### *Simple Variables*

To access any of the simple variables, we use the id of the group (1.3.6.1.2.1.7) followed by the id of the variable. The following shows how to access each variable.

| | | |
|---|---|---|
| **udpInDatagrams** | → | **1.3.6.1.2.1.7.1** |
| **udpNoPorts** | → | **1.3.6.1.2.1.7.2** |
| **udpInErrors** | → | **1.3.6.1.2.1.7.3** |
| **udpOutDatagrams** | → | **1.3.6.1.2.1.7.4** |

However, these object identifiers define the variable, not the instance (contents). To show the instance or the contents of each variable, we must add an instance suffix. The instance suffix for a simple variable is simply a zero. In other words, to show an instance of the above variables, we use the following:

| | | |
|---|---|---|
| **udpInDatagrams.0** | → | **1.3.6.1.2.1.7.1.0** |
| **udpNoPorts.0** | → | **1.3.6.1.2.1.7.2.0** |
| **udpInErrors.0** | → | **1.3.6.1.2.1.7.3.0** |
| **udpOutDatagrams.0** | → | **1.3.6.1.2.1.7.4.0** |

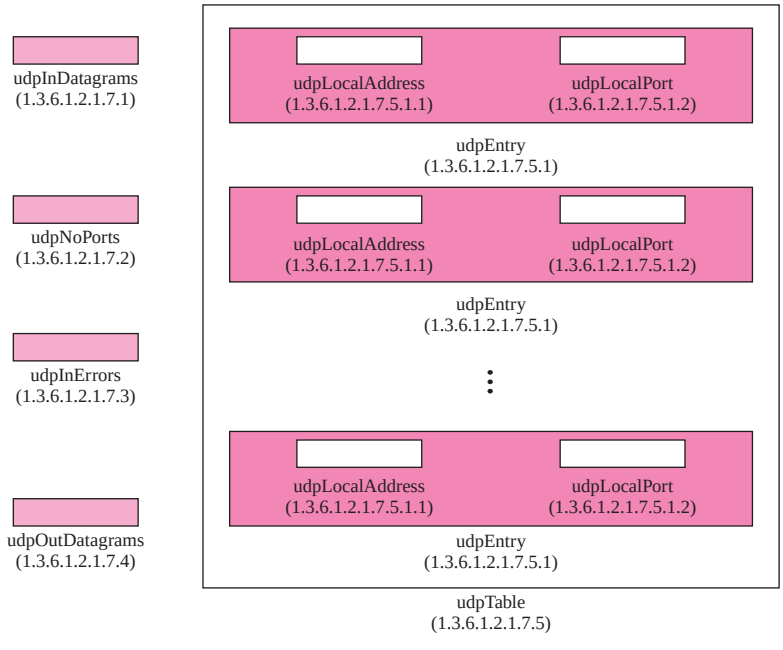
### *Tables*

To identify a table, we first use the table id. The udp group has only one table (with id 5), as illustrated in Figure 24.14.

So to access the table, we use the following:

| | | |
|---|---|---|
| **udpTable** | → | **1.3.6.1.2.1.7.5** |

**Figure 24.14** *udp variables and tables*




However, the table is not at the leaf level in the tree structure. We cannot access the table; we define the entry (sequence) in the table (with id of 1), as follows:

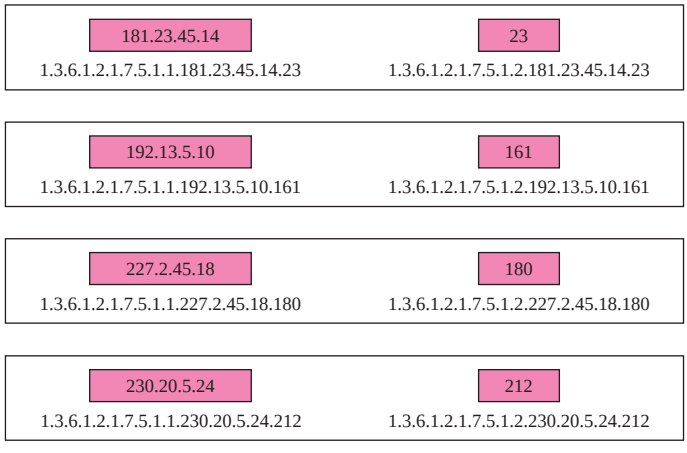
| | | |
|---|---|---|
| **udpEntry** | → | **1.3.6.1.2.1.7.5.1** |

This entry is also not a leaf and we cannot access it. We need to define each entity (field) in the entry.

| | | |
|---|---|---|
| **udpLocalAddress** | → | **1.3.6.1.2.1.7.5.1.1** |
| **udpLocalPort** | → | **1.3.6.1.2.1.7.5.1.2** |

These two variables are at the leaf of the tree. Although we can access their instances, we need to define *which* instance. At any moment, the table can have several values for each local address/local port pair. To access a specific instance (row) of the table, we add the index to the above ids. In MIB, the indexes of arrays are not integers (like most programming languages). The indexes are based on the value of one or more fields in the entries. In our example, the udpTable is indexed based on both the local address and the local port number. For example, Figure 24.15 shows a table with four rows and values for each field. The index of each row is a combination of two values.

**Figure 24.15** *Indexes for udpTable*




To access the instance of the local address for the first row, we use the identifier augmented with the instance index:

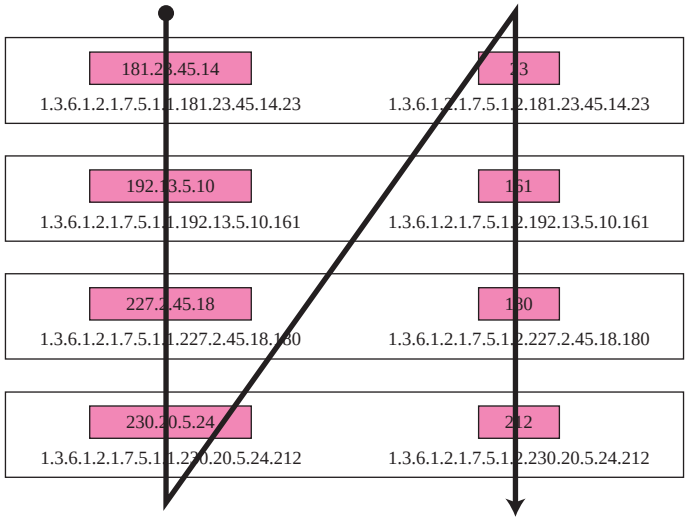
| | | |
|---|---|---|
| **udpLocalAddress.181.23.45.14.23** | → | **1.3.6.1.2.7.5.1.1.181.23.45.14.2** |

Note that not all tables are indexed the same way. Some tables are indexed using the value of one field, some using the value of two fields, and so on.

## Lexicographic Ordering

One interesting point about the MIB variables is that the object identifiers (including the instance identifiers) follow in lexicographic order. Tables are ordered according to column-row rules, which means one should go column by column. In each column, one should go from the top to the bottom, as shown in Figure 24.16.

**Figure 24.16** *Lexicographic ordering*




The **lexicographic ordering** enables a manager to access a set of variables one after another by defining the first variable, as we will see in the GetNextRequest command in the next section.

## 24.5 SNMP

SNMP uses both SMI and MIB in Internet network management. It is an application program that allows:

1. A manager to retrieve the value of an object defined in an agent.
2. A manager to store a value in an object defined in an agent.
3. An agent to send an alarm message about an abnormal situation to the manager.

### PDUs

SNMPv3 defines eight types of protocol data units (or PDUs): GetRequest, GetNextRequest, GetBulkRequest, SetRequest, Response, Trap, InformRequest, and Report (see Figure 24.17).

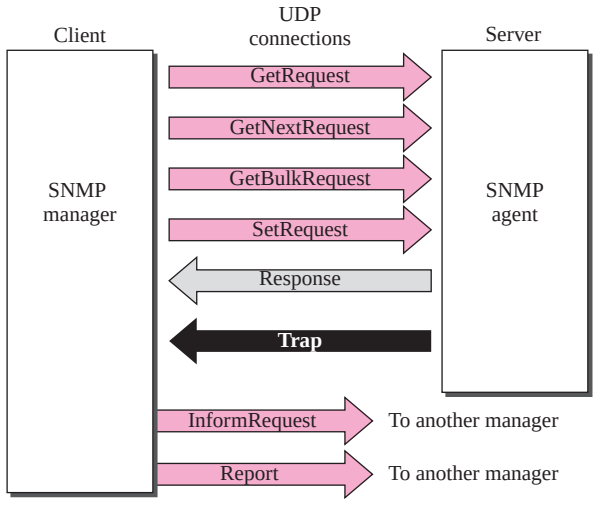
#### *GetRequest*

The GetRequest PDU is sent from the manager (client) to the agent (server) to retrieve the value of a variable or a set of variables.

#### *GetNextRequest*

The GetNextRequest PDU is sent from the manager to the agent to retrieve the value of a variable. The retrieved value is the value of the object following the defined ObjectId in the PDU. It is mostly used to retrieve the values of the entries in a table. If the manager does not know the indexes of the entries, it cannot retrieve the values. However, it can use GetNextRequest and define the ObjectId of the table. Because the first entry has

**Figure 24.17** *SNMP PDUs*




the ObjectId immediately after the ObjectId of the table, the value of the first entry is returned. The manager can use this ObjectId to get the value of the next one, and so on.

### *GetBulkRequest*

The GetBulkRequest PDU is sent from the manager to the agent to retrieve a large amount of data. It can be used instead of multiple GetRequest and GetNextRequest PDUs.

### *SetRequest*

The SetRequest PDU is sent from the manager to the agent to set (store) a value in a variable.

### *Response*

The Response PDU is sent from an agent to a manager in response to GetRequest or GetNextRequest. It contains the value(s) of the variable(s) requested by the manager.

### *Trap*

The **Trap** (also called SNMPv2 Trap to distinguish it from SNMPv1 Trap) PDU is sent from the agent to the manager to report an event. For example, if the agent is rebooted, it informs the manager and reports the time of rebooting.

### *InformRequest*

The InformRequest PDU is sent from one manager to another remote manager to get the value of some variables from agents under the control of the remote manager. The remote manager responds with a Response PDU.

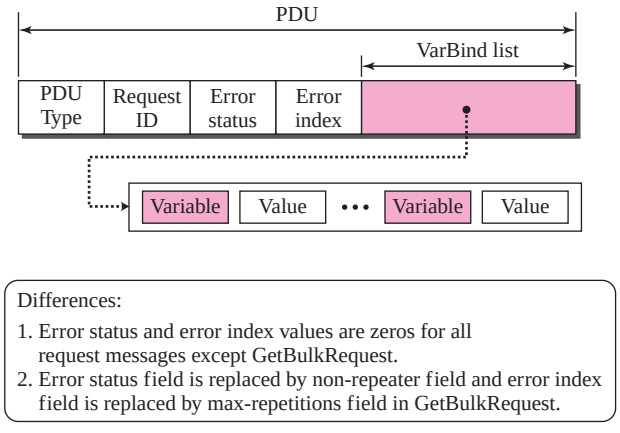
### *Report*

The Report PDU is designed to report some types of errors between managers. It is not yet in use.

## Format

The format for the eight SNMP PDUs is shown in Figure 24.18. The GetBulkRequest PDU differs from the others in two areas as shown in the figure.

**Figure 24.18** *SNMP PDU format*




The fields are listed below:

- **PDU type.** This field defines the type of the PDU (see Table 24.3) .

**Table 24.3** *PDU Types*

| *Type* | *Tag (Binary)* | *Tag (Hex)* |
|---|---|---|
| GetRequest | **10100000** | **A0** |
| GetNextRequest | **10100001** | **A1** |
| Response | **10100010** | **A2** |
| SetRequest | **10100011** | **A3** |
| GetBulkRequest | **10100101** | **A5** |
| InformRequest | **10100110** | **A6** |
| Trap (SNMPv2) | **10100111** | **A7** |
| Report | **10101000** | **A8** |

- **Request ID.** This field is a sequence number used by the manager in a request PDU and repeated by the agent in a response. It is used to match a request to a response.
- **Error status.** This is an integer that is used only in response PDUs to show the types of errors reported by the agent. Its value is 0 in request PDUs. Table 24.4 lists the types of errors that can occur.
- **Non-repeaters.** This field is used only in GetBulkRequest and replaces the error status field, which is empty in request PDUs.
- **Error index.** The error index is an offset that tells the manager which variable caused the error.

**Table 24.4** *Types of Errors*

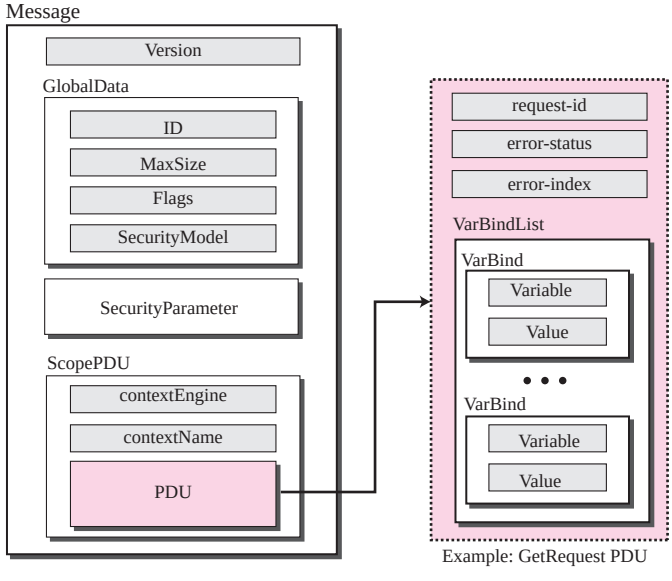
| *Status* | *Name* | *Meaning* |
|---|---|---|
| 0 | noError | No error |
| 1 | tooBig | Response too big to fit in one message |
| 2 | noSuchName | Variable does not exist |
| 3 | badValue | The value to be stored is invalid |
| 4 | readOnly | The value cannot be modified |
| 5 | genErr | Other errors |

- **Max-repetition.** This field is also used only in GetBulkRequest and replaces the error index field, which is empty in request PDUs.
- **VarBind list.** This is a set of variables with the corresponding values the manager wants to retrieve or set. The values are null in GetRequest and GetNextRequest. In a Trap PDU, it shows the variables and values related to a specific PDU.

## Messages

SNMP does not send only a PDU, it embeds the PDU in a message. A message in SNMPv3 is a sequence made of four elements: Version, GlobalData, SecurityParameters, and ScopePDU (which includes the encoded PDU) as shown in Figure 24.19. The first and the third elements are simple data types; the second and the fourth are sequences.

**Figure 24.19** *SNMP message*




### *Version*

The Version field is an INTEGER data type that defines the version. The current version is 3.

### *GlobalData*

The GlobalData field is a sequence with four elements of simple data type: ID, MaxSize, Flags, and SecurityModel.

### *Security Parameter*

This element is a sequence that can be very complex, depending on the type of security provision used in version 3.

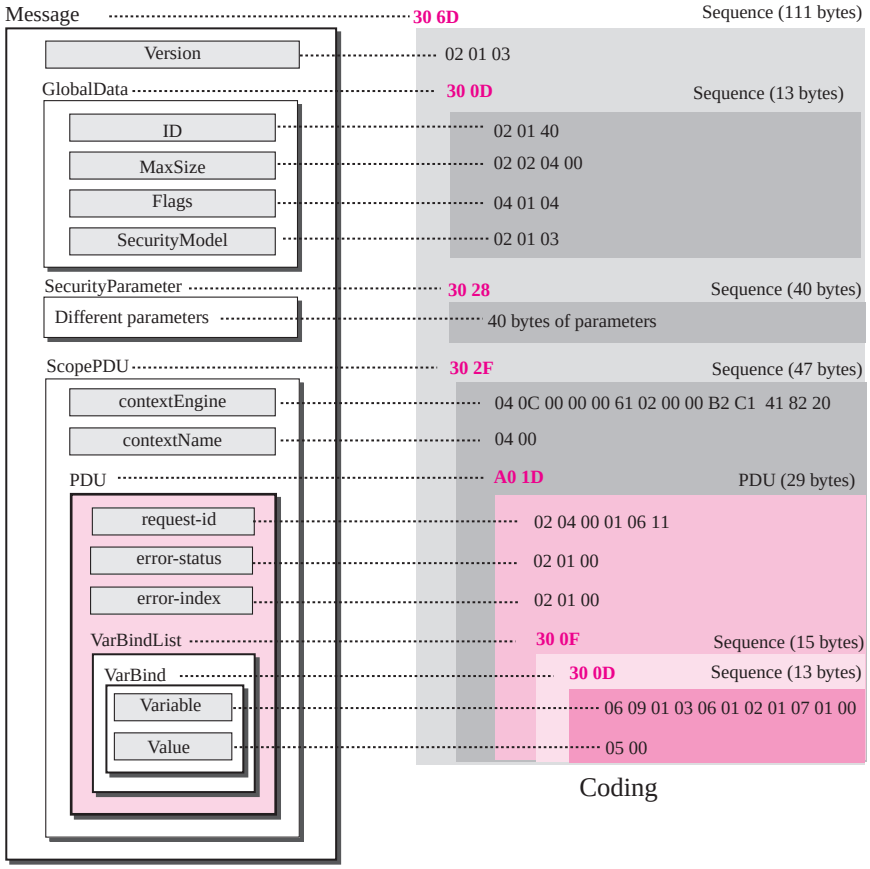
### *ScopePDU*

The last element contains two simple data type and the actual PDU. We have shown only one example of GetRequest PDU. Note that VarBindList is a sequence made of one or more sequences called VarBind. Each VarBind is made of two simple data elements: Variable and Value.

### Example 24.5

In this example, a manager station (SNMP client) uses a message with GetRequest PDU to retrieve the number of UDP datagrams that a router has received (Figure 24.20). There is only

**Figure 24.20** *Example 24.5*



one VarBind sequence. The corresponding MIB variable related to this information is udpInDatagrams with the object identifier 1.3.6.1.2.1.7.1.0. The manager wants to retrieve a value (not to store a value), so the value defines a null entity. The bytes to be sent are shown in hexadecimal representation.

The VarBind list has only one VarBind. The variable is of type 06 and length 09. The value is of type 05 and length 00. The whole VarBind is a sequence of length 0D (13). The VarBind list is also a sequence of length 0F (15). The GetRequest PDU is of length ID (29). The PDU is embedded in ScopePDU sequence, which is of 47 bytes. The Security Parameters assumed to be 40 bytes, but the details are not shown. The GlobalData itself is a sequence of 13 bytes. Three sequences and one integer (version) are embedded in the message sequence, which is the length of 111 bytes. The whole message is of 113 bytes.

Note that we have intended the bytes to show the inclusion of simple data types inside a sequence or the inclusion of sequences and simple data type inside larger sequences. Note that the PDU itself is like a sequence, but its tag is A0 in hexadecimal.

Figure 24.21 shows the actual message sent. We show the message using rows of 16 bytes to save space on the paper, but the actual message is sent in rows of 4 bytes. The bytes that are shown using dashes are the one related to the security parameters.

**Figure 24.21** *Actual message sent for Example 24.5*

Message

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **30** | **6D** | 02 | 01 | 03 | **30** | **0D** | 02 | 01 | 04 | 02 | 02 | 04 | 00 | 04 | 01 |
| 04 | 02 | 01 | 03 | **30** | **28** | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- |
| -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- |
| -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | **30** | **2F** |
| 04 | 0C | 00 | 00 | 00 | 61 | 02 | 00 | 00 | B2 | C1 | 41 | 82 | 20 | 04 | 00 |
| **A0** | **1D** | 02 | 04 | 00 | 01 | 06 | 11 | 02 | 01 | 00 | 02 | 01 | 00 | **30** | **0F** |
| **30** | **0D** | 06 | 09 | 01 | 03 | 06 | 01 | 02 | 01 | 07 | 01 | 00 | 05 | 00 | |

## 24.6 UDP PORTS

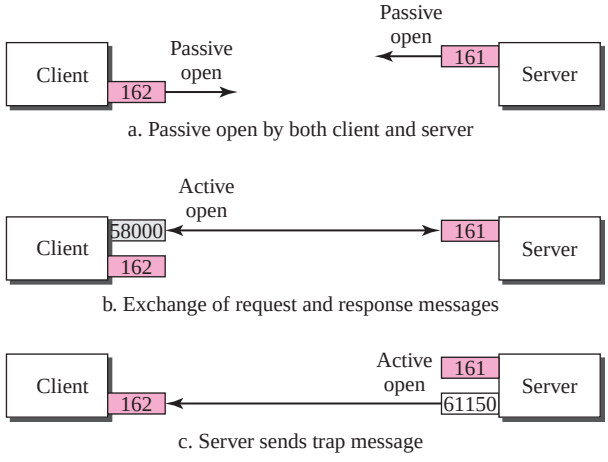
SNMP uses the services of UDP on two well-known ports, 161 and 162. The well-known port 161 is used by the server (agent), and the well-known port 162 is used by the client (manager).

The agent (server) issues a passive open on port 161. It then waits for a connection from a manager (client). A manager (client) issues an active open using an ephemeral port. The request messages are sent from the client to the server using the ephemeral port as the source port and the well-known port 161 as the destination port. The response messages are sent from the server to the client using the well-known port 161 as the source port and the ephemeral port as the destination port.

The manager (client) issues a passive open on port 162. It then waits for a connection from an agent (server). Whenever it has a Trap message to send, an agent (server) issues an active open, using an ephemeral port. This connection is only one-way, from the server to the client (see Figure 24.22).

The client-server mechanism in SNMP is different from other protocols. Here both the client and the server use well-known ports. In addition, both the client and the

**Figure 24.22** *Port numbers for SNMP*




server are running infinitely. The reason is that request messages are initiated by a manager (client), but Trap messages are initiated by an agent (server).

## 24.7 SECURITY

SNMPv3 has added two new features to the previous version: security and remote administration. SNMPv3 allows a manager to choose one or more levels of security when accessing an agent. Different aspects of security can be configured by the manager to allow message authentication, confidentiality, and integrity.

SNMPv3 also allows remote configuration of security aspects without requiring the administrator to actually be at the place where the device is located.

## 24.8 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of SNMP [Com 06], [Ste 94], and [Tan 03], and [Kur & Ros 08]. [Mau & Sch 01] is a book totally devoted to SNMP and recommended for deep study of different features present in this protocol.

### RFCs

Several RFCs show different update on SMTP including RFC 3410, RFC 3412, RFC 3415, and RFC 3418. More information about MIB can be found in RFC 2578, RFC 2579, and RFC 2580.

## 24.9 KEY TERMS

Abstract Syntax Notation 1 (ASN.1)
agent
Basic Encoding Rules (BER)
lexicographic ordering
Management Information Base (MIB)
manager
object identifier
simple data type
Simple Network Management Protocol (SNMP)
Structure of Management Information (SMI)
structured data type
Trap

## 24.10 SUMMARY

- Simple Network Management Protocol (SNMP) is a framework for managing devices in an internet using the TCP/IP protocol suite. A manager, usually a host, controls and monitors a set of agents, usually routers. The manager is a host that runs the SNMP client program. The agent is a router or host that runs the SNMP server program. SNMP frees management tasks from both the physical characteristics of the managed devices and the underlying networking technology. SNMP uses the services of two other protocols: Structure of Management Information (SMI) and Management Information Base (MIB).
- SMI names objects, defines the type of data that can be stored in an object, and encodes the data. SMI objects are named according to a hierarchical tree structure. SMI data types are defined according to Abstract Syntax Notation 1 (ASN.1). SMI uses Basic Encoding Rules (BER) to encode data.
- MIB is a collection of groups of objects that can be managed by SNMP. MIB uses lexicographic ordering to manage its variables.
- SNMP functions in three ways: A manager can retrieve the value of an object defined in an agent. A manager can store a value in an object defined in an agent. An agent can send an alarm message to the manager.
- SNMP defines eight types of packets: GetRequest, GetNextRequest, SetRequest, GetBulkRequest, Trap, InformRequest, Response, and Report. SNMP uses the services of UDP on two well-known ports, 161 and 162. SNMPv3 has enhanced security features over previous versions.
- The third version of SNMP has added two new features to the previous version: different levels of security and remote administration.

## 24.11 PRACTICE SET

### Exercises

1. Show the encoding for the INTEGER 1456.
2. Show the encoding for the OCTET STRING “Hello World.”

3. Show the encoding for an arbitrary OCTET STRING of length 1,000.
4. Show how the following record (sequence) is encoded.

| INTEGER | OCTET STRING | IP Address |
|---|---|---|
| 2345 | "COMPUTER" | 185.32.1.5 |

5. Show how the following record (sequence) is encoded.

| Time Tick | INTEGER | Object Id |
|---|---|---|
| 12000 | 14564 | 1.3.6.1.2.1.7 |

6. Show how the following array (sequence) is encoded. Each element is an integer.

| | | | |
|---|---|---|---|
| 2345 | 1236 | 122 | 1236 |

7. Show how the following array of records (sequence of sequences) is encoded using the BER encoding described in the chapter. Note that the titles for each column

| INTEGER | OCTET STRING | Counter |
|---|---|---|
| 2345 | "COMPUTER" | 345 |
| 1123 | "DISK" | 1430 |
| 3456 | "MONITOR" | 2313 |

shows the type of the data item.

8. Decode the following:
   a. 02 04 01 02 14 32
   b. 30 06 02 01 11 02 01 14
   c. 30 09 04 03 41 43 42 02 02 14 14
   d. 30 0A 40 04 23 51 62 71 02 02 14 12

## Research Activity

9. Find more information about ASN.1.

CHAPTER

# 25 Multimedia

Recent advances in technology have changed our use of audio and video. In the past, we listened to an audio broadcast through a radio and watched a video program broadcast through a TV. We used the telephone network to interactively communicate with another party. But times have changed. People want to use the Internet not only for text and image communications, but also for audio and video services. In this chapter, we concentrate on applications that use the Internet for audio and video services.

## OBJECTIVES

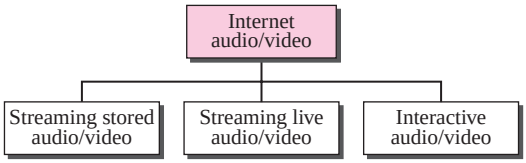
*The chapter has several objectives:*

- To show how audio/video files can be downloaded for future use or broadcast to clients over the Internet. The Internet can also be used for live audio/video interaction. Audio and video need to be digitized before being sent over the Internet.
- To discuss how audio and video files are compressed for transmission through the Internet.
- To discuss the phenomenon called Jitter that can be created on a packet-switched network when transmitting real-time data.
- To introduce the Real-Time Transport Protocol (RTP) and Real-Time Transport Control Protocol (RTCP) used in multimedia applications.
- To discuss voice over IP as a real-time interactive audio/video application.
- To introduce the Session Initiation Protocol (SIP) as an application layer protocol that establishes, manages, and terminates multimedia sessions.
- To introduce quality of service (QoS) and how it can be improved using scheduling techniques and traffic shaping techniques.
- To discuss Integrated Services and Differential Services and how they can be implemented.
- To introduce Resource Reservation Protocol (RSVP) as a signaling protocol that helps IP create a flow and makes a resource reservation.

## 25.1 INTRODUCTION

We can divide audio and video services into three broad categories: **streaming stored audio/video, streaming live audio/video,** and **interactive audio/video,** as shown in Figure 25.1. Streaming means a user can listen (or watch) the file after the downloading has started.

**Figure 25.1** *Internet audio/video*




In the first category, streaming stored audio/video, the files are compressed and stored on a server. A client downloads the files through the Internet. This is sometimes referred to as **on-demand audio/video.** Examples of stored audio files are songs, symphonies, books on tape, and famous lectures. Examples of stored video files are movies, TV shows, and music video clips.

**Streaming stored audio/video refers to on-demand requests for compressed audio/video files.**

In the second category, streaming live audio/video, a user listens to broadcast audio and video through the Internet. A good example of this type of application is the Internet radio. Some radio stations broadcast their programs only on the Internet; many broadcast them both on the Internet and on the air. Internet TV is not popular yet, but many people believe that TV stations will broadcast their programs on the Internet in the future.

**Streaming live audio/video refers to the broadcasting of radio and TV programs through the Internet.**

In the third category, interactive audio/video, people use the Internet to interactively communicate with one another. A good example of this application is Internet telephony and Internet teleconferencing.

**Interactive audio/video refers to the use of the Internet for interactive audio/video applications.**

We will discuss these three applications in this chapter, but first we need to discuss some other issues related to audio/video: digitizing audio and video and compressing audio and video.

## 25.2 DIGITIZING AUDIO AND VIDEO

Before audio or video signals can be sent on the Internet, they need to be digitized. We discuss audio and video separately.

### Digitizing Audio

When sound is fed into a microphone, an electronic analog signal is generated that represents the sound amplitude as a function of time. The signal is called an *analog audio signal*. An analog signal, such as audio, can be digitized to produce a digital signal. According to the Nyquist theorem, if the highest frequency of the signal is $f$, we need to sample the signal $2f$ times per second. There are other methods for digitizing an audio signal, but the principle is the same.

Voice is sampled at 8,000 samples per second with 8 bits per sample. This results in a digital signal of 64 kbps. Music is sampled at 44,100 samples per second with 16 bits per sample. This results in a digital signal of 705.6 kbps for monaural and 1.411 Mbps for stereo.

### Digitizing Video

A video consists of a sequence of frames. If the frames are displayed on the screen fast enough, we get an impression of motion. The reason is that our eyes cannot distinguish the rapidly flashing frames as individual ones. There is no standard number of frames per second; in North America 25 frames per second is common. However, to avoid a condition known as flickering, a frame needs to be refreshed. The TV industry repaints each frame twice. This means 50 frames need to be sent, or if there is memory at the sender site, 25 frames with each frame repainted from the memory.

Each frame is divided into small grids, called picture elements or **pixels.** For black-and-white TV, each 8-bit pixel represents one of 256 different gray levels. For a color TV, each pixel is 24 bits, with 8 bits for each primary color (red, green, and blue).

We can calculate the number of bits in a second for a specific resolution. In the lowest resolution a color frame is made of $1,024 \times 768$ pixels. This means that we need

$$2 \times 25 \times 1,024 \times 768 \times 24 = 944 \text{ Mbps}$$

This data rate needs a very high data rate technology such as SONET. To send video using lower-rate technologies, we need to compress the video.

**Compression is needed to send video over the Internet.**

## 25.3 AUDIO AND VIDEO COMPRESSION

To send audio or video over the Internet requires **compression.** In this section, we first discuss audio compression and then video compression.

### Audio Compression

Audio compression can be used for speech or music. For speech, we need to compress a 64-kHz digitized signal; for music, we need to compress a 1.411-MHz signal. Two categories of techniques are used for audio compression: predictive encoding and perceptual encoding.

#### *Predictive Encoding*

In **predictive encoding,** the differences between the samples are encoded instead of encoding all the sampled values. This type of compression is normally used for speech. Several standards have been defined such as GSM (13 kbps), G.729 (8 kbps), and G.723.3 (6.4 or 5.3 kbps). Detailed discussions of these techniques are beyond the scope of this book.

#### *Perceptual Encoding: MP3*

The most common compression technique that is used to create CD-quality audio is based on the **perceptual encoding** technique. As we mentioned before, this type of audio needs at least 1.411 Mbps; this cannot be sent over the Internet without compression. **MP3** (MPEG audio layer 3), a part of the MPEG standard (discussed in the video compression section), uses this technique.

Perceptual encoding is based on the science of psychoacoustics, which is the study of how people perceive sound. The idea is based on some flaws in our auditory system: Some sounds can mask other sounds. Masking can happen in frequency and time. In **frequency masking,** a loud sound in a frequency range can partially or totally mask a softer sound in another frequency range. For example, we cannot hear what our dance partner says in a room where a loud heavy metal band is performing. In **temporal masking,** a loud sound can numb our ears for a short time even after the sound has stopped.

MP3 uses these two phenomena, frequency and temporal masking, to compress audio signals. The technique analyzes and divides the spectrum into several groups. Zero bits are allocated to the frequency ranges that are totally masked. A small number of bits are allocated to the frequency ranges that are partially masked. A larger number of bits are allocated to the frequency ranges that are not masked.

MP3 produces three data rates: 96 kbps, 128 kbps, and 160 kbps. The rate is based on the range of the frequencies in the original analog audio.

### Video Compression

As we mentioned before, video is composed of multiple frames. Each frame is one image. We can compress video by first compressing images. Two standards are prevalent in the market. **Joint Photographic Experts Group (JPEG)** is used to compress

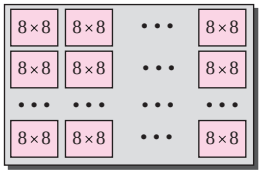
images. **Moving Picture Experts Group (MPEG)** is used to compress video. We briefly discuss JPEG and then MPEG.

### *Image Compression: JPEG*

As we discussed previously, if the picture is not in color (gray scale), each pixel can be represented by an 8-bit integer (256 levels). If the picture is in color, each pixel can be represented by 24 bits ($3 \times 8$ bits), with each 8 bits representing red, blue, or green (RBG). To simplify the discussion, we concentrate on a gray scale picture.

In JPEG, a gray scale picture is divided into blocks of $8 \times 8$ pixels (see Figure 25.2).

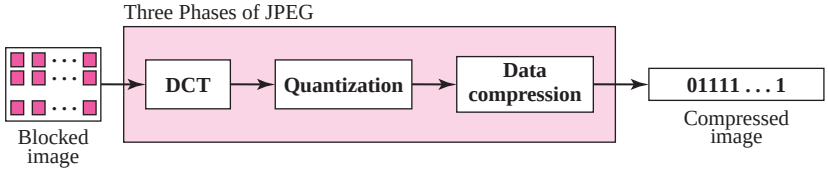
**Figure 25.2** *JPEG gray scale*




The purpose of dividing the picture into blocks is to decrease the number of calculations because, as you will see shortly, the number of mathematical operations for each picture is the square of the number of units.

The whole idea of JPEG is to change the picture into a linear (vector) set of numbers that reveals the redundancies. The redundancies (lack of changes) can then be removed by using one of the text compression methods. A simplified scheme of the process is shown in Figure 25.3.

**Figure 25.3** *JPEG process*

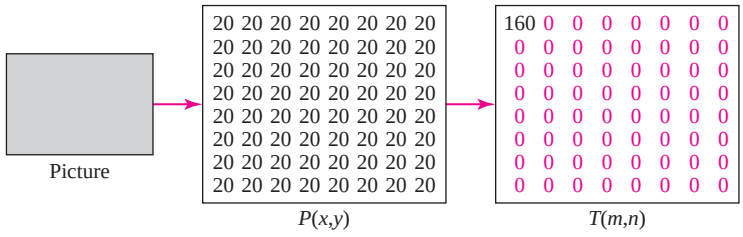



**Discrete Cosine Transform (DCT)** In this step, each block of 64 pixels goes through a transformation called the **discrete cosine transform (DCT).** The transformation changes the 64 values so that the relative relationships between pixels are kept but the redundancies are revealed. We do not give the formula here, but we do show the results of the transformation for three cases.

**Case 1** In this case, we have a block of uniform gray, and the value of each pixel is 20. When we do the transformations, we get a nonzero value for the first element

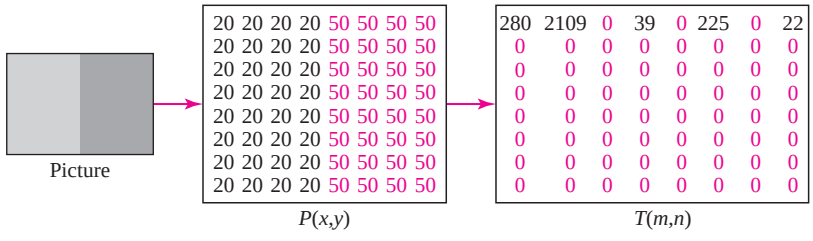
(upper left corner); the rest of the pixels have a value of 0. The value of $T(0,0)$ is the average (multiplied by a constant) of the $P(x,y)$ values and is called the *dc value* (direct current, borrowed from electrical engineering). The rest of the values, called *ac values*, in $T(m,n)$ represent changes in the pixel values. But because there are no changes, the rest of the values are 0s (see Figure 25.4).

**Figure 25.4** *Case 1: uniform gray scale*




**Case 2** In the second case, we have a block with two different uniform gray scale sections. There is a sharp change in the values of the pixels (from 20 to 50). When we do the transformations, we get a dc value as well as nonzero ac values. However, there are only a few nonzero values clustered around the dc value. Most of the values are 0 (see Figure 25.5).

**Figure 25.5** *Case 2: two sections*

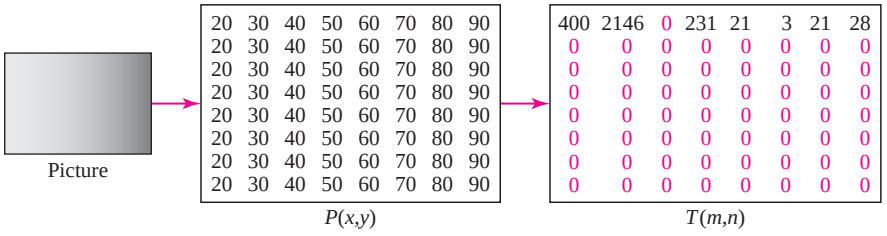



**Case 3** In the third case, we have a block that changes gradually. That is, there is no sharp change between the values of neighboring pixels. When we do the transformations, we get a dc value, with many nonzero ac values also (Figure 25.6).

We can say the following:

- The transformation creates table *T* from table *P*.
- The dc value is the average value (multiplied by a constant) of the pixels.
- The ac values are the changes.
- Lack of changes in neighboring pixels creates 0s.

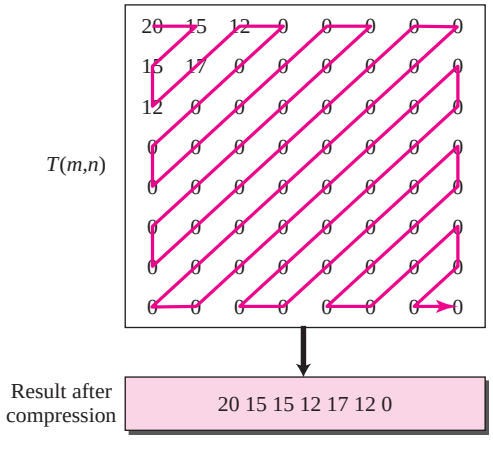
**Figure 25.6** *Case 3: gradient gray scale*




**Quantization** After the *T* table is created, the values are quantized to reduce the number of bits needed for encoding. Previously in **quantization,** we dropped the fraction from each value and kept the integer part. Here, we divide the number by a constant and then drop the fraction. This reduces the required number of bits even more. In most implementations, a quantizing table (8 by 8) defines how to quantize each value. The divisor depends on the position of the value in the *T* table. This is done to optimize the number of bits and the number of 0s for each particular application. Note that the only phase in the process that is not completely reversible is the quantizing phase. We lose some information here that is not recoverable. As a matter of fact, the only reason that JPEG is called *lossy compression* is because of this quantization phase.

**Compression** After quantization, the values are read from the table, and redundant 0s are removed. However, to cluster the 0s together, the table is read diagonally in a zigzag fashion rather than row by row or column by column. The reason is that if the picture changes smoothly, the bottom right corner of the *T* table is all 0s. Figure 25.7 shows the process.

**Figure 25.7** *Reading the table*



### *Video Compression: MPEG*

The Moving Picture Experts Group (MPEG) method is used to compress video. In principle, a motion picture is a rapid flow of a set of frames, where each frame is an image. In other words, a frame is a spatial combination of pixels, and a video is a temporal combination of frames that are sent one after another. Compressing video, then, means spatially compressing each frame and temporally compressing a set of frames.

**Spatial Compression** The **spatial compression** of each frame is done with JPEG (or a modification of it). Each frame is a picture that can be independently compressed.

**Temporal Compression** In **temporal compression,** redundant frames are removed. When we watch television, we receive 50 frames per second. However, most of the consecutive frames are almost the same. For example, when someone is talking, most of the frame is the same as the previous one except for the segment of the frame around the lips, which changes from one frame to another.

To temporally compress data, the MPEG method first divides frames into three categories: I-frames, P-frames, and B-frames. Figure 25.8 shows a sample sequence of frames.

**Figure 25.8** *MPEG frames*

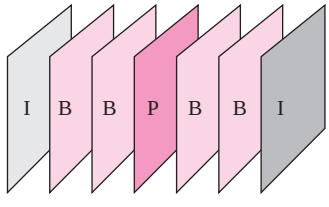
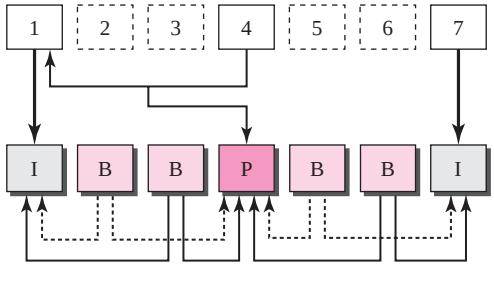



Figure 25.9 shows how I-, P-, and B-frames are constructed from a series of seven frames.

**Figure 25.9** *MPEG frame construction*



- **I-frames.** An **intracoded frame (I-frame)** is an independent frame that is not related to any other frame (not to the frame sent before or to the frame sent after). They are present at regular intervals (e.g., every ninth frame is an I-frame). An I-frame must appear periodically to handle some sudden change in the frame that the previous and following frames cannot show. Also, when a video is broadcast, a viewer may tune at any time. If there is only one I-frame at the beginning of the broadcast, the viewer who tunes in late will not receive a complete picture. I-frames are independent of other frames and cannot be constructed from other frames.
- **P-frames.** A **predicted frame (P-frame)** is related to the preceding I-frame or P-frame. In other words, each P-frame contains only the changes from the preceding frame. The changes, however, cannot cover a big segment. For example, for a fast-moving object, the new changes may not be recorded in a P-frame. P-frames can be constructed only from previous I- or P-frames. P-frames carry much less information than other frame types and carry even fewer bits after compression.
- **B-frames.** A **bidirectional frame (B-frame)** is related to the preceding and following I-frame or P-frame. In other words, each B-frame is relative to the past and the future. Note that a B-frame is never related to another B-frame.

MPEG has gone through two versions. MPEG1 was designed for a CD-ROM with a data rate of 1.5 Mbps. MPEG2 was designed for high-quality DVD with a data rate of 3 to 6 Mbps.

## 25.4 STREAMING STORED AUDIO/VIDEO

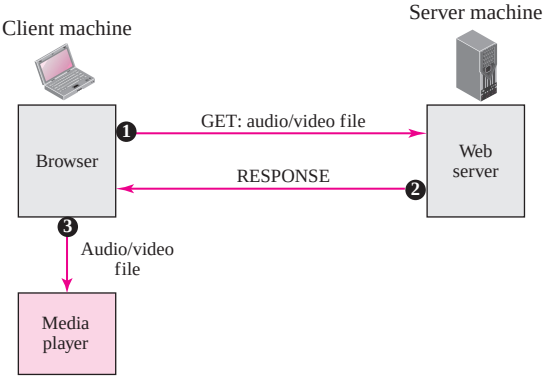
Now that we have discussed digitizing and compressing audio/video, we turn our attention to specific applications. The first is streaming stored audio and video. Downloading these types of files from a Web server can be different from downloading other types of files. To understand the concept, let us discuss three approaches, each with a different complexity.

### First Approach: Using a Web Server

A compressed audio/video file can be downloaded as a text file. The client (browser) can use the services of HTTP and send a GET message to download the file. The Web server can send the compressed file to the browser. The browser can then use a help application, normally called a **media player,** to play the file. Figure 25.10 shows this approach.

This approach is very simple and does not involve *streaming*. However, it has a drawback. An audio/video file is usually large even after compression. An audio file may contain tens of megabits, and a video file may contain hundreds of megabits. In this approach, the file needs to download completely before it can be played. Using contemporary data rates, the user needs some seconds or tens of seconds before the file can be played.

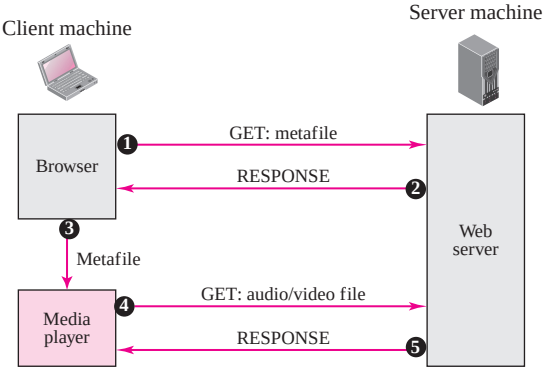
**Figure 25.10** *Using a Web server*




## Second Approach: Using a Web Server with Metafile

In another approach, the media player is directly connected to the Web server for downloading the audio/video file. The Web server stores two files: the actual audio/video file and a **metafile** that holds information about the audio/video file. Figure 25.11 shows the steps in this approach.

**Figure 25.11** *Using a Web server with a metafile*

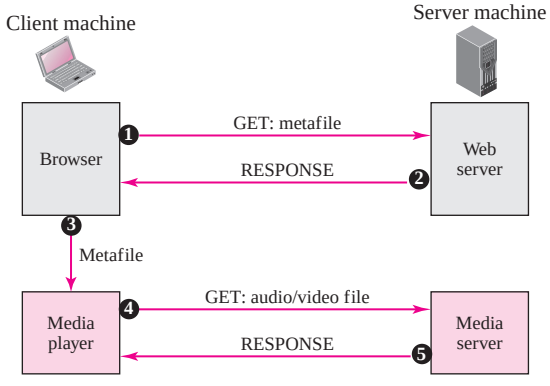



1. The HTTP client accesses the Web server using the GET message.
2. The information about the metafile comes in the response.
3. The metafile is passed to the media player.
4. The media player uses the URL in the metafile to access the audio/video file.
5. The Web server responds.

## Third Approach: Using a Media Server

The problem with the second approach is that the browser and the media player both use the services of HTTP. HTTP is designed to run over TCP. This is appropriate for retrieving the metafile, but not for retrieving the audio/video file. The reason is that TCP retransmits a lost or damaged segment, which is counter to the philosophy of streaming. We need to dismiss TCP and its error control; we need to use UDP. However, HTTP, which accesses the Web server, and the Web server itself are designed for TCP; we need another server, a **media server.** Figure 25.12 shows the concept.

**Figure 25.12** *Using a media server*



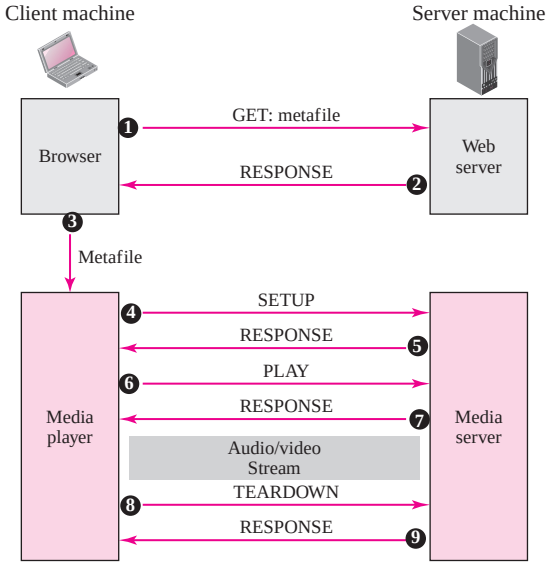

1. The HTTP client accesses the Web server using a GET message.
2. The information about the metafile comes in the response.
3. The metafile is passed to the media player.
4. The media player uses the URL in the metafile to access the media server to download the file. Downloading can take place by any protocol that uses UDP.
5. The media server responds.

## Fourth Approach: Using a Media Server and RTSP

The **Real-Time Streaming Protocol (RTSP)** is a control protocol designed to add more functionalities to the streaming process. Using RTSP, we can control the playing of audio/video. RTSP is an out-of-band control protocol that is similar to the second connection in FTP. Figure 25.13 shows a media server and RTSP.

1. The HTTP client accesses the Web server using a GET message.
2. The information about the metafile comes in the response.
3. The metafile is passed to the media player.
4. The media player sends a SETUP message to create a connection with the media server.

**Figure 25.13** *Using a media server and RTSP*




5. The media server responds.
6. The media player sends a PLAY message to start playing (downloading).
7. The audio/video file is downloaded using another protocol that runs over UDP.
8. The connection is broken using the TEARDOWN message.
9. The media server responds.

The media player can send other types of messages. For example, a PAUSE message temporarily stops the downloading; downloading can be resumed with a PLAY message.

## 25.5 STREAMING LIVE AUDIO/VIDEO

Streaming live audio/video is similar to the broadcasting of audio and video by radio and TV stations. Instead of broadcasting to the air, the stations broadcast through the Internet. There are several similarities between streaming stored audio/video and streaming live audio/video. They are both sensitive to delay; neither can accept retransmission. However, there is a difference. In the first application, the communication is unicast and on-demand. In the second, the communication is multicast and live. Live streaming is better suited to the multicast services of IP and the use of protocols such as UDP and RTP (discussed later). However, presently, live streaming is still using TCP and multiple unicasting instead of multicasting. There is still much progress to be made in this area.

## 25.6 REAL-TIME INTERACTIVE AUDIO/VIDEO

In real-time interactive audio/video, people communicate with one another in real time. The Internet phone or voice over IP is an example of this type of application. Video conferencing is another example that allows people to communicate visually and orally.

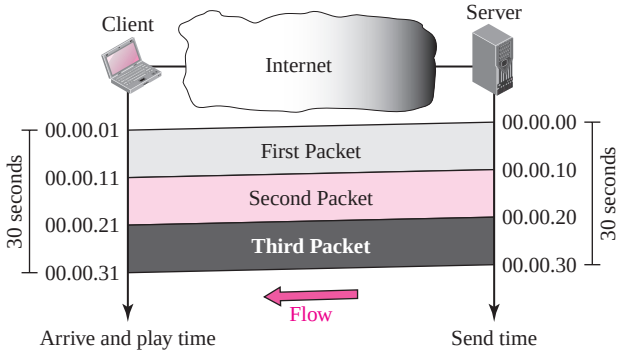
### Characteristics

Before discussing the protocols used in this class of applications, we discuss some characteristics of real-time audio/video communication.

#### *Time Relationship*

Real-time data on a packet-switched network require the preservation of the time relationship between packets of a session. For example, let us assume that a real-time video server creates live video images and sends them online. The video is digitized and packetized. There are only three packets, and each packet holds 10 s of video information. The first packet starts at 00:00:00, the second packet starts at 00:00:10, and the third packet starts at 00:00:20. Also imagine that it takes 1 s (an exaggeration for simplicity) for each packet to reach the destination (equal delay). The receiver can play back the first packet at 00:00:01, the second packet at 00:00:11, and the third packet at 00:00:21. Although there is a 1-s time difference between what the server sends and what the client sees on the computer screen, the action is happening in real time. The time relationship between the packets is preserved. The 1-s delay is not important. Figure 25.14 shows the idea.

**Figure 25.14** *Time relationship*

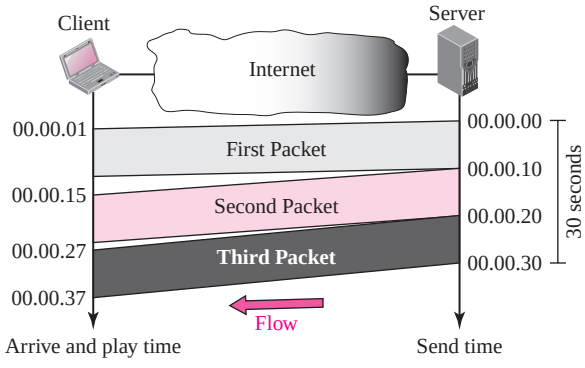



But what happens if the packets arrive with different delays? For example, the first packet arrives at 00:00:01 (1-s delay), the second arrives at 00:00:15 (5-s delay), and the third arrives at 00:00:27 (7-s delay). If the receiver starts playing the first packet at 00:00:01, it will finish at 00:00:11. However, the next packet has not yet arrived; it

arrives 4 s later. There is a gap between the first and second packets and between the second and the third as the video is viewed at the remote site. This phenomenon is called **jitter.** Figure 25.15 shows the situation.

**Jitter is introduced in real-time data by the delay between packets.**

**Figure 25.15** *Jitter*




### *Timestamp*

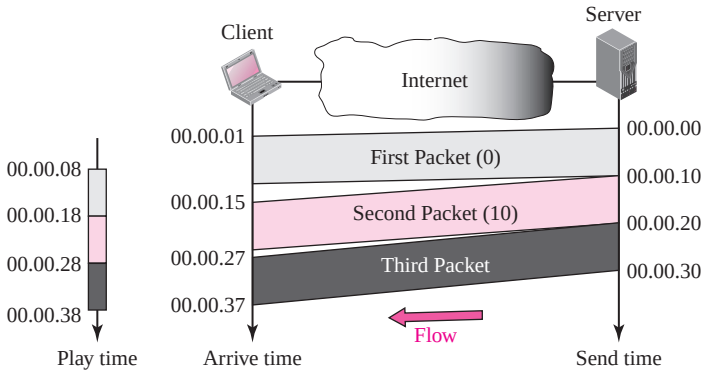
One solution to jitter is the use of a **timestamp.** If each packet has a timestamp that shows the time it was produced relative to the first (or previous) packet, then the receiver can add this time to the time at which it starts the playback. In other words, the receiver knows when each packet is to be played. Imagine the first packet in the previous example has a timestamp of 0, the second has a timestamp of 10, and the third a timestamp of 20. If the receiver starts playing back the first packet at 00:00:08, the second will be played at 00:00:18, and the third at 00:00:28. There are no gaps between the packets. Figure 25.16 shows the situation.

### *Playback Buffer*

To be able to separate the arrival time from the playback time, we need a buffer to store the data until they are played back. The buffer is referred to as a **playback buffer.** When a session begins (the first bit of the first packet arrives), the receiver delays playing the data until a threshold is reached. In the previous example, the first bit of the first packet arrives at 00:00:01; the threshold is 7 s, and the playback time is 00:00:08. The threshold is measured in time units of data. The replay does not start until the time units of data are equal to the threshold value.

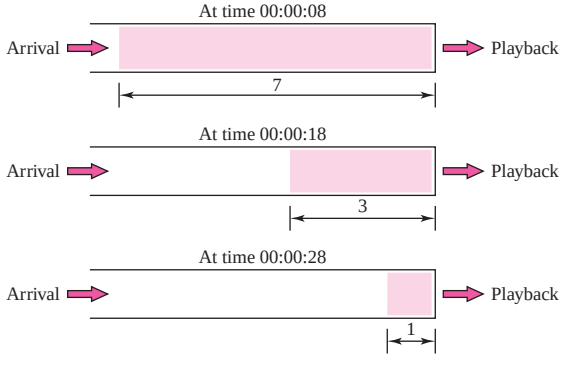
Data are stored in the buffer at a possibly variable rate, but they are extracted and played back at a fixed rate. Note that the amount of data in the buffer shrinks or expands, but as long as the delay is less than the time to play back the threshold amount of data, there is no jitter. Figure 25.17 shows the buffer at different times for our example.

**Figure 25.16** *Timestamp*




**To prevent jitter, we can timestamp the packets and separate the arrival time from the playback time.**

**Figure 25.17** *Playback buffer*




**A playback buffer is required for real-time traffic.**

### *Ordering*

In addition to time relationship information and timestamps for real-time traffic, one more feature is needed. We need a *sequence number* for each packet. The timestamp alone cannot inform the receiver if a packet is lost. For example, suppose the timestamps are 0, 10, and 20. If the second packet is lost, the receiver receives just two packets with timestamps 0 and 20. The receiver assumes that the packet with timestamp 20 is the

second packet, produced 20 s after the first. The receiver has no way of knowing that the second packet has actually been lost. A sequence number to order the packets is needed to handle this situation.

**A sequence number on each packet is required for real-time traffic.**

### *Multicasting*

Multimedia play a primary role in audio and video conferencing. The traffic can be heavy, and the data are distributed using **multicasting** methods. Conferencing requires two-way communication between receivers and senders.

**Real-time traffic needs the support of multicasting.**

### *Translation*

Sometimes real-time traffic needs **translation.** A translator is a computer that can change the format of a high-bandwidth video signal to a lower-quality narrow-bandwidth signal. This is needed, for example, for a source creating a high-quality video signal at 5 Mbps and sending to a recipient having a bandwidth of less than 1 Mbps. To receive the signal, a translator is needed to decode the signal and encode it again at a lower quality that needs less bandwidth.

**Translation means changing the encoding of a payload to a lower quality to match the bandwidth of the receiving network.**

### *Mixing*

If there is more than one source that can send data at the same time (as in a video or audio conference), the traffic is made of multiple streams. To converge the traffic to one stream, data from different sources can be mixed. A **mixer** mathematically adds signals coming from different sources to create one single signal.

**Mixing means combining several streams of traffic into one stream.**

### *Support from Transport Layer Protocol*

The procedures mentioned in the previous sections can be implemented in the application layer. However, they are so common in real-time applications that implementation in the transport layer protocol is preferable. Let's see which of the existing transport layers is suitable for this type of traffic.

TCP is not suitable for interactive traffic. It has no provision for timestamping, and it does not support multicasting. However, it does provide ordering (sequence numbers). One feature of TCP that makes it particularly unsuitable for interactive traffic is its error control mechanism. In interactive traffic, we cannot allow the retransmission of

a lost or corrupted packet. If a packet is lost or corrupted in interactive traffic, it must just be ignored. Retransmission upsets the whole idea of timestamping and playback. Today there is so much redundancy in audio and video signals (even with compression) that we can simply ignore a lost packet. The listener or viewer at the remote site may not even notice it.

**TCP, with all its sophistication, is not suitable for interactive multimedia traffic because we cannot allow retransmission of packets.**

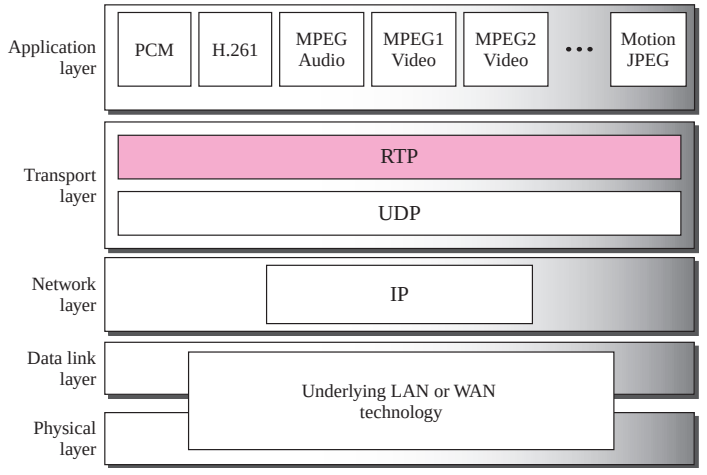
UDP is more suitable for interactive multimedia traffic. UDP supports multicasting and has no retransmission strategy. However, UDP has no provision for timestamping, sequencing, or mixing. A new transport protocol, Real-Time Transport Protocol (RTP), provides these missing features.

**UDP is more suitable than TCP for interactive traffic. However, we need the services of RTP, another transport layer protocol, to make up for the deficiencies of UDP.**

## 25.7 RTP

**Real-time Transport Protocol (RTP)** is the protocol designed to handle real-time traffic on the Internet. RTP does not have a delivery mechanism (multicasting, port numbers, and so on); it must be used with UDP. RTP stands between UDP and the application program. The main contributions of RTP are timestamping, sequencing, and mixing facilities. Figure 25.18 shows the position of RTP in the protocol suite.

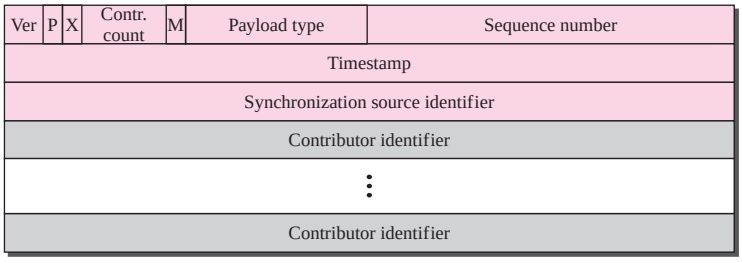
**Figure 25.18** *RTP*



## RTP Packet Format

Figure 25.19 shows the format of the RTP packet header. The format is very simple and general enough to cover all real-time applications. An application that needs more information adds it to the beginning of its payload. A description of each field follows.

**Figure 25.19** *RTP packet header format*




- **Ver.** This 2-bit field defines the version number. The current version is 2.
- **P.** This 1-bit field, if set to 1, indicates the presence of padding at the end of the packet. In this case, the value of the last byte in the padding defines the length of the padding. Padding is the norm if a packet is encrypted. There is no padding if the value of the P field is 0.
- **X.** This 1-bit field, if set to 1, indicates an extra extension header between the basic header and the data. There is no extra extension header if the value of this field is 0.
- **Contributor count.** This 4-bit field indicates the number of contributors. Note that we can have a maximum of 15 contributors because a 4-bit field only allows a number between 0 and 15.
- **M.** This 1-bit field is a marker used by the application to indicate, for example, the end of its data.
- **Payload type.** This 7-bit field indicates the type of the payload. Several payload types have been defined so far. We list some common applications in Table 25.1. A discussion of the types is beyond the scope of this book.

**Table 25.1** *Payload Types*

| *Type* | *Application* | *Type* | *Application* | *Type* | *Application* |
|---|---|---|---|---|---|
| 0 | PCMμ Audio | 7 | LPC audio | 15 | G728 audio |
| 1 | 1016 | 8 | PCMA audio | 26 | Motion JPEG |
| 2 | G721 audio | 9 | G722 audio | 31 | H.261 |
| 3 | GSM audio | 10–11 | L16 audio | 32 | MPEG1 video |
| 5–6 | DV14 audio | 14 | MPEG audio | 33 | MPEG2 video |

- **Sequence number.** This field is 16 bits in length. It is used to number the RTP packets. The sequence number of the first packet is chosen randomly; it is incremented by 1 for each subsequent packet. The sequence number is used by the receiver to detect lost or out of order packets.
- **Timestamp.** This is a 32-bit field that indicates the time relationship between packets. The timestamp for the first packet is a random number. For each succeeding packet, the value is the sum of the preceding timestamp plus the time the first byte is produced (sampled). The value of the clock tick depends on the application. For example, audio applications normally generate chunks of 160 bytes; the clock tick for this application is 160. The timestamp for this application increases 160 for each RTP packet.
- **Synchronization source identifier.** If there is only one source, this 32-bit field defines the source. However, if there are several sources, the mixer is the synchronization source and the other sources are contributors. The value of the source identifier is a random number chosen by the source. The protocol provides a strategy in case of conflict (two sources start with the same sequence number).
- **Contributor identifier.** Each of these 32-bit identifiers (a maximum of 15) defines a source. When there is more than one source in a session, the mixer is the synchronization source and the remaining sources are the contributors.

### UDP Port

Although RTP is itself a transport layer protocol, the RTP packet is not encapsulated directly in an IP datagram. Instead, RTP is treated like an application program and is encapsulated in a UDP user datagram. However, unlike other application programs, no well-known port is assigned to RTP. The port can be selected on demand with only one restriction: The port number must be an even number. The next number (an odd number) is used by the companion of RTP, Real-Time Transport Control Protocol (RTCP).

**RTP uses a temporary even-numbered UDP port.**

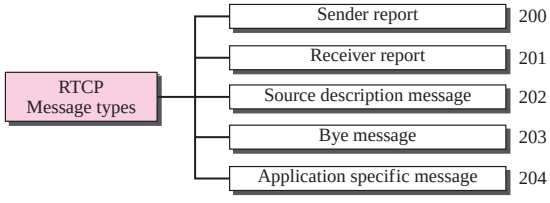
## 25.8 RTCP

RTP allows only one type of message, one that carries data from the source to the destination. In many cases, there is a need for other messages in a session. These messages control the flow and quality of data and allow the recipient to send feedback to the source or sources. **Real-Time Transport Control Protocol (RTCP)** is a protocol designed for this purpose. RTCP has five types of messages, as shown in Figure 25.20. The number next to each box defines the type of the message.

### Sender Report

The sender report is sent periodically by the active senders in a conference to report transmission and reception statistics for all RTP packets sent during the interval. The

**Figure 25.20** *RTCP message types*




sender report includes an absolute timestamp, which is the number of seconds elapsed since midnight January 1, 1970. The absolute timestamp allows the receiver to synchronize different RTP messages. It is particularly important when both audio and video are transmitted (audio and video transmissions use separate relative timestamps).

## Receiver Report

The receiver report is for passive participants, those that do not send RTP packets. The report informs the sender and other receivers about the quality of service.

## Source Description Message

The source periodically sends a source description message to give additional information about itself. This information can be the name, e-mail address, telephone number, and address of the owner or controller of the source.

## Bye Message

A source sends a bye message to shut down a stream. It allows the source to announce that it is leaving the conference. Although other sources can detect the absence of a source, this message is a direct announcement. It is also very useful to a mixer.

## Application-Specific Message

The application-specific message is a packet for an application that wants to use new applications (not defined in the standard). It allows the definition of a new message type.

## UDP Port

RTCP, like RTP, does not use a well-known UDP port. It uses a temporary port. The UDP port chosen must be the number immediately following the UDP port selected for RTP, which makes it an odd-numbered port.

**RTCP uses an odd-numbered UDP port number that follows the port number selected for RTP.**

## 25.9 VOICE OVER IP

Let us concentrate on one real-time interactive audio/video application: **voice over IP,** or Internet telephony. The idea is to use the Internet as a telephone network with some additional capabilities. Instead of communicating over a circuit-switched network, this application allows communication between two parties over the packet-switched Internet. Two protocols have been designed to handle this type of communication: SIP and H.323. We briefly discuss both.

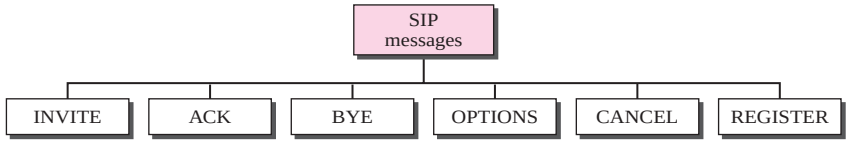
### SIP

The **Session Initiation Protocol (SIP)** was designed by IETF. It is an application layer protocol that establishes, manages, and terminates a multimedia session (call). It can be used to create two-party, multiparty, or multicast sessions. SIP is designed to be independent of the underlying transport layer; it can run on either UDP, TCP, or SCTP.

#### *Messages*

SIP is a text-based protocol like HTTP. SIP, like HTTP, uses messages. Six messages are defined as shown in Figure 25.21.

**Figure 25.21** *SIP messages*




Each message has a header and a body. The header consists of several lines that describe the structure of the message, caller's capability, media type, and so on. We give a brief description of each message. Then we show their applications in a simple session.

The caller initializes a session with the INVITE message. After the callee answers the call, the caller sends an ACK message for confirmation. The BYE message terminates a session. The OPTIONS message queries a machine about its capabilities. The CANCEL message cancels an already started initialization process. The REGISTER message makes a connection when the callee is not available.

#### *Addresses*

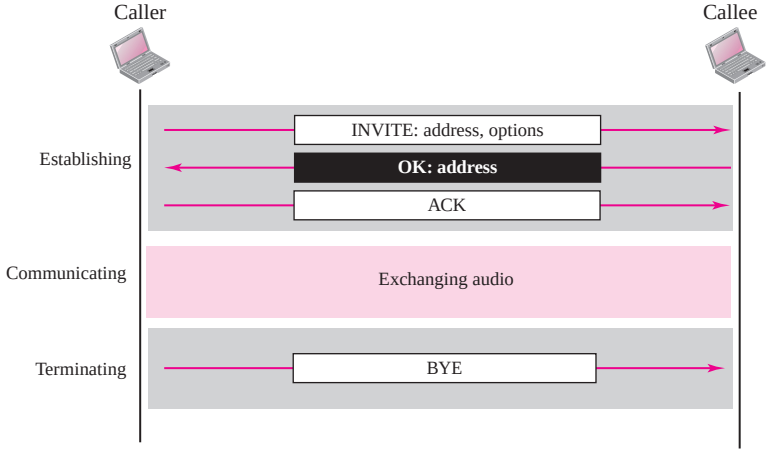
In a regular telephone communication a telephone number identifies the sender, and another telephone number identifies the receiver. SIP is very flexible. In SIP, an e-mail address, an IP address, a telephone number, and other types of addresses can be used to identify the sender and receiver. However, the address needs to be in SIP format (also called scheme). Figure 25.22 shows some common formats.

**Figure 25.22** *SIP formats*


### *Simple Session*

A simple session using SIP consists of three modules: establishing, communicating, and terminating. Figure 25.23 shows a simple session using SIP.

**Figure 25.23** *SIP simple session*




**Establishing a Session** Establishing a session in SIP requires a three-way handshake. The caller sends an INVITE message, using UDP, TCP, or SCTP to begin the communication. If the callee is willing to start the session, she sends a reply message. To confirm that a reply code has been received, the caller sends an ACK message.

**Communicating** After the session has been established, the caller and the callee can communicate using two temporary ports.

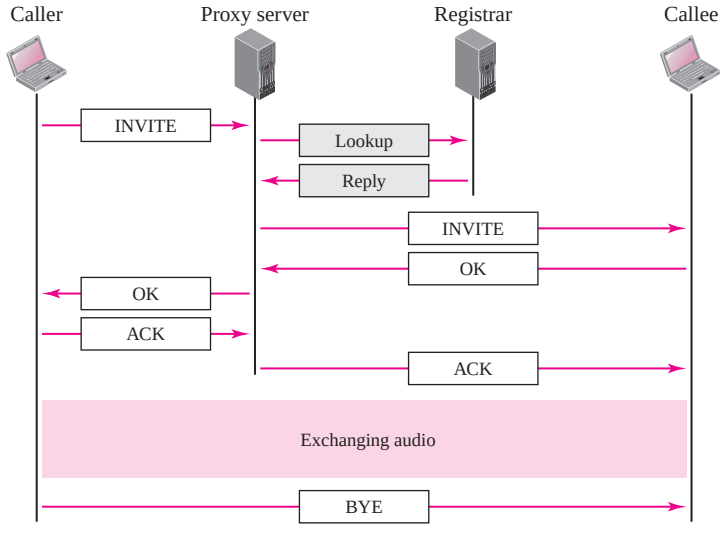
**Terminating the Session** The session can be terminated with a BYE message sent by either party.

### *Tracking the Callee*

What happens if the callee is not sitting at her terminal? She may be away from her system or at another terminal. She may not even have a fixed IP address if DHCP is being used. SIP has a mechanism (similar to one in DNS) that finds the IP address of the terminal at which the callee is sitting. To perform this tracking, SIP uses the concept of registration. SIP defines some servers as registrars. At any moment a user is registered with at least one **registrar server;** this server knows the IP address of the callee.

When a caller needs to communicate with the callee, the caller can use the e-mail address instead of the IP address in the INVITE message. The message goes to a proxy server. The proxy server sends a lookup message (not part of SIP) to some registrar server that has registered the callee. When the proxy server receives a reply message from the registrar server, the proxy server takes the caller's INVITE message and inserts the newly discovered IP address of the callee. This message is then sent to the callee. Figure 25.24 shows the process.

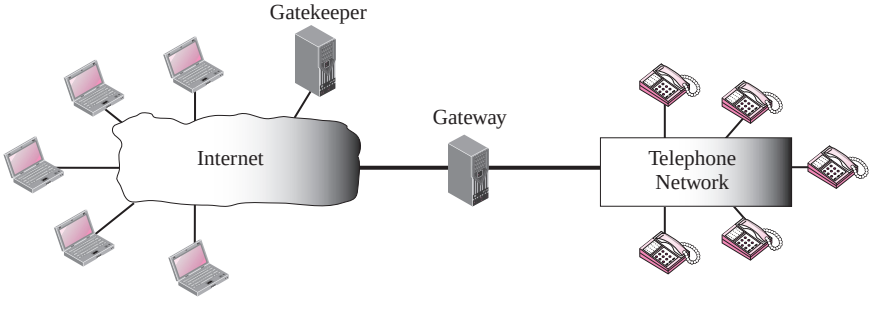
**Figure 25.24** *Tracking the callee*




## H.323

**H.323** is a standard designed by ITU to allow telephones on the public telephone network to talk to computers (called *terminals* in H.323) connected to the Internet. Figure 25.25 shows the general architecture of H.323.

**Figure 25.25** *H.323 architecture*

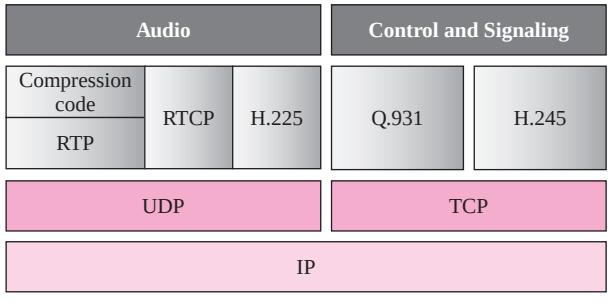


A **gateway** connects the Internet to the telephone network. In general, a gateway is a five-layer device that can translate a message from one protocol stack to another. The gateway here does exactly the same thing. It transforms a telephone network message into an Internet message. The **gatekeeper** server on the local area network plays the role of the registrar server, as we discussed in the SIP protocol.

### *Protocols*

H.323 uses a number of protocols to establish and maintain voice (or video) communication. Figure 25.26 shows these protocols.

**Figure 25.26** *H.323 protocols*



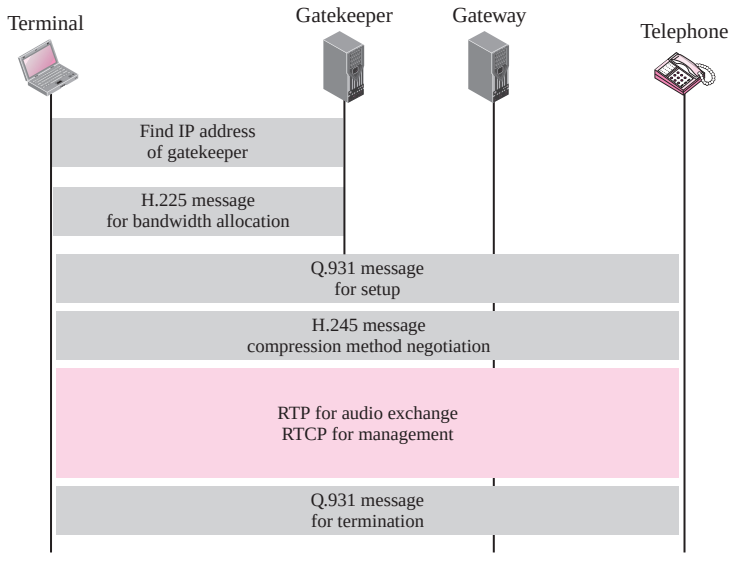

H.323 uses G.71 or G.723.1 for compression. It uses a protocol named H.245 which allows the parties to negotiate the compression method. Protocol Q.931 is used for establishing and terminating connections. Another protocol called H.225, or RAS (Registration/Administration/Status), is used for registration with the gatekeeper.

### *Operation*

Let us show the operation of a telephone communication using H.323 with a simple example. Figure 25.27 shows the steps used by a terminal to communicate with a telephone.

1. The terminal sends a broadcast message to the gatekeeper. The gatekeeper responds with its IP address.
2. The terminal and gatekeeper communicate, using H.225 to negotiate bandwidth.
3. The terminal, the gatekeeper, gateway, and the telephone communicate using Q.931 to set up a connection.
4. The terminal, the gatekeeper, gateway, and the telephone communicate using H.245 to negotiate the compression method.
5. The terminal, gateway, and the telephone exchange audio using RTP under the management of RTCP.
6. The terminal, the gatekeeper, gateway, and the telephone communicate using Q.931 to terminate the communication.

**Figure 25.27** *H.323 example*



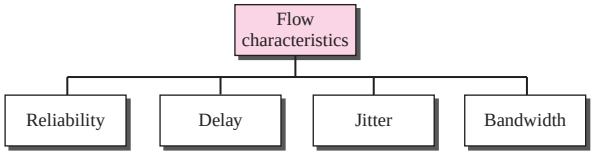

## 25.10 QUALITY OF SERVICE

**Quality of service (QoS)** is an internetworking issue that has been discussed more than defined. We can informally define quality of service as something a flow of data seeks to attain. Although QoS can be applied to both textual data and multimedia, it is more an issue when we are dealing with multimedia.

### Flow Characteristics

Traditionally, four types of characteristics are attributed to a flow: reliability, delay, jitter, and bandwidth, as shown in Figure 25.28.

**Figure 25.28** *Flow characteristics*




#### *Reliability*

**Reliability** is a characteristic that a flow needs. Lack of reliability means losing a packet or acknowledgment, which entails retransmission. However, the sensitivity of

application programs to reliability is not the same. For example, it is more important that electronic mail, file transfer, and Internet access have reliable transmissions than telephony or audio conferencing.

### *Delay*

Source-to-destination **delay** is another flow characteristic. Again applications can tolerate delay in different degrees. In this case, telephony, audio conferencing, video conferencing, and remote log-in need minimum delay, while delay in file transfer or e-mail is less important.

### *Jitter*

**Jitter** is the variation in delay for packets belonging to the same flow. For example, if four packets depart at times 0, 1, 2, 3 and arrive at 20, 21, 22, 23, all have the same delay, 20 units of time. On the other hand, if the above four packets arrive at 21, 23, 21, and 28, they will have different delays: 21, 22, 19, and 24.

For applications such as audio and video, the first case is completely acceptable; the second case is not. For these applications, it does not matter if the packets arrive with a short or long delay as long as the delay is the same for all packets. For this application, the second case is not acceptable.

Jitter is defined as the variation in the packet delay. High jitter means the difference between delays is large; low jitter means the variation is small.

### *Bandwidth*

Different applications need different bandwidths. In video conferencing we need to send millions of bits per second to refresh a color screen while the total number of bits in an e-mail may not reach even a million.

## Flow Classes

Based on the flow characteristics, we can classify flows into groups, with each group having similar levels of characteristics. This categorization is not formal or universal; some protocols such as ATM have defined classes.

## Techniques to Improve QoS

In the previous section we tried to define QoS in terms of its characteristics. In this section, we discuss some techniques that can be used to improve the quality of service. We briefly discuss four common methods: scheduling, traffic shaping, admission control, and resource reservation.

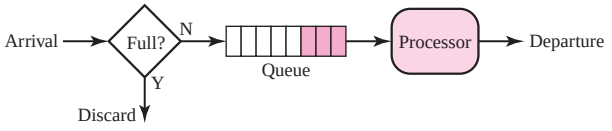
### *Scheduling*

Packets from different flows arrive at a switch or router for processing. A good scheduling technique treats the different flows in a fair and appropriate manner. Several scheduling techniques are designed to improve the quality of service. We discuss three of them here: FIFO queuing, priority queuing, and weighted fair queuing.

**FIFO Queuing** In **first-in, first-out (FIFO) queuing,** packets wait in a buffer (queue) until the node (router or switch) is ready to process them. If the average arrival

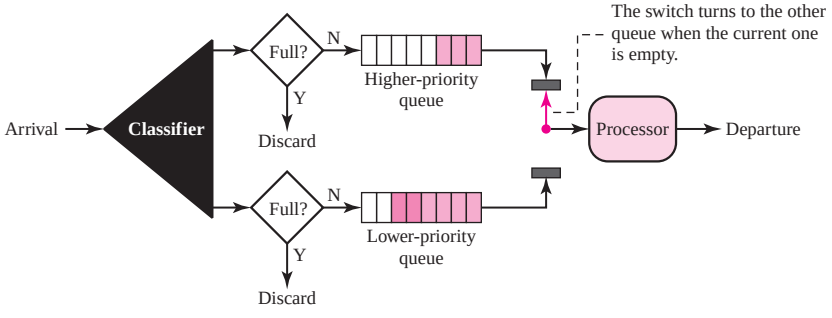
rate is higher than the average processing rate, the queue will fill up and new packets will be discarded. A FIFO queue is familiar to those who have had to wait for a bus at a bus stop. Figure 25.29 shows a conceptual view of a FIFO queue.

**Figure 25.29** *FIFO queue*




**Priority Queuing** In **priority queuing,** packets are first assigned to a priority class. Each priority class has its own queue. The packets in the highest-priority queue are processed first. Packets in the lowest-priority queue are processed last. Note that the system does not stop serving a queue until it is empty. Figure 25.30 shows priority queuing with two priority levels (for simplicity).

**Figure 25.30** *Priority queuing*

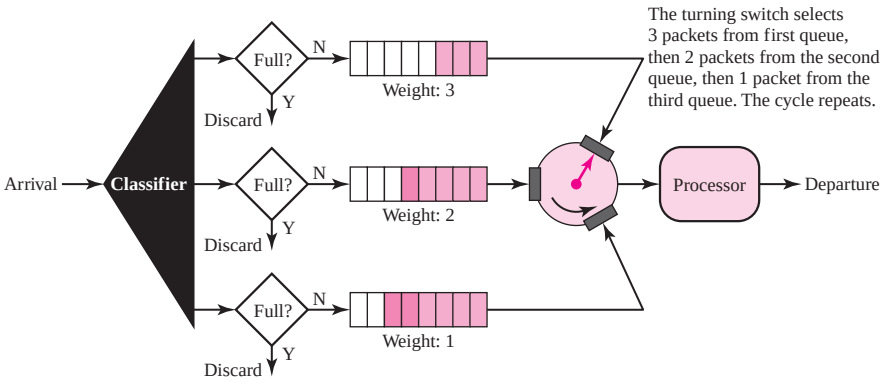



A priority queue can provide better QoS than the FIFO queue because higher-priority traffic, such as multimedia, can reach the destination with less delay. However, there is a potential drawback. If there is a continuous flow in a high-priority queue, the packets in the lower-priority queues will never have a chance to be processed. This is a condition called *starvation*.

**Weighted Fair Queuing** A better scheduling method is **weighted fair queuing.** In this technique, the packets are still assigned to different classes and admitted to different queues. The queues, however, are weighted based on the priority of the queues; higher priority means a higher weight. The system processes packets in each queue in a round-robin fashion with the number of packets selected from each queue based on the

corresponding weight. For example, if the weights are 3, 2, and 1, three packets are processed from the first queue, two from the second queue, and one from the third queue. If the system does not impose priority on the classes, all weights can be equal. In this way, we have fair queuing with priority. Figure 25.31 shows the technique with three classes.

**Figure 25.31** *Weighted fair queuing*



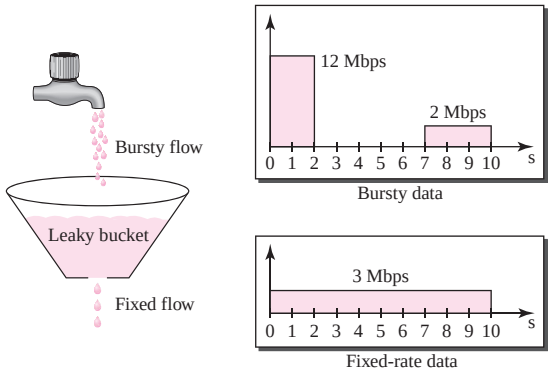

## *Traffic Shaping*

**Traffic shaping** is a mechanism to control the amount and the rate of the traffic sent to the network. Two techniques can shape traffic: leaky bucket and token bucket.

**Leaky Bucket** If a bucket has a small hole at the bottom, the water leaks from the bucket at a constant rate as long as there is water in the bucket. The rate at which the water leaks does not depend on the rate at which the water is input to the bucket unless the bucket is empty. The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called **leaky bucket** can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate. Figure 25.32 shows a leaky bucket and its effects.

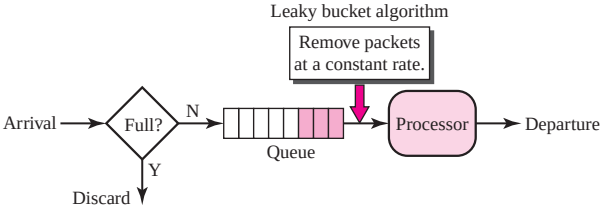
In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment. In Figure 25.32 the host sends a burst of data at a rate of 12 Mbps for 2 s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in 10 s. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10 s. Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion. As an analogy, consider the freeway during rush hour (bursty traffic). If, instead, commuters could stagger their working hours, congestion on our freeways could be avoided.

**Figure 25.32** *Leaky bucket*




A simple leaky bucket implementation is shown in Figure 25.33. A FIFO queue holds the packets. If the traffic consists of fixed-size packets (e.g., cells in ATM networks), the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.

**Figure 25.33** *Leaky bucket implementation*



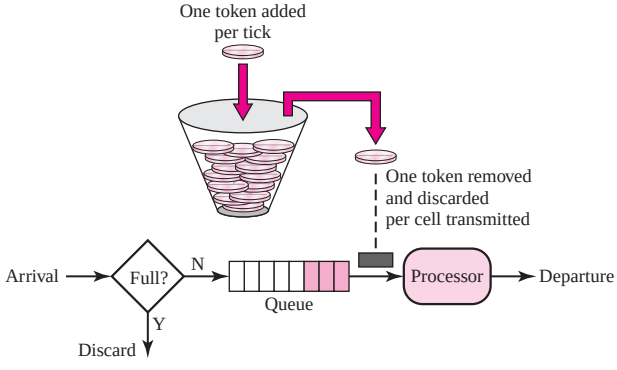

The following is an algorithm for variable-length packets:

1. Initialize a counter to *n* at the tick of the clock.
2. If *n* is greater than the size of the packet, send the packet and decrement the counter by the packet size. Repeat this step until *n* is smaller than the packet size.
3. Reset the counter and go to step 1.

**A leaky bucket algorithm shapes bursty traffic into fixed-rate traffic by averaging the data rate. It may drop the packets if the bucket is full.**

**Token Bucket** The leaky bucket is very restrictive. It does not credit an idle host. For example, if a host is not sending for a while, its bucket becomes empty. Now if the host has bursty data, the leaky bucket allows only an average rate. The time when the host was idle is not taken into account. On the other hand, the **token bucket** algorithm allows idle hosts to accumulate credit for the future in the form of tokens. For each tick of the clock, the system sends *n* tokens to the bucket. The system removes one token for every cell (or byte) of data sent. For example, if *n* is 100 and the host is idle for 100 ticks, the bucket collects 10,000 tokens. Now the host can consume all these tokens in one tick with 10,000 cells, or the host takes 1,000 ticks with 10 cells per tick. In other words, the host can send bursty data as long as the bucket is not empty. Figure 25.34 shows the idea.

**Figure 25.34** *Token bucket*




The token bucket can easily be implemented with a counter. The token is initialized to zero. Each time a token is added, the counter is incremented by 1. Each time a unit of data is sent, the counter is decremented by 1. When the counter is zero, the host cannot send data.

**The token bucket allows bursty traffic at a regulated maximum rate.**

**Combining Token Bucket and Leaky Bucket** The two techniques can be combined to credit an idle host and at the same time regulate the traffic. The leaky bucket is applied after the token bucket; the rate of the leaky bucket needs to be higher than the rate of tokens dropped in the bucket.

## Resource Reservation

A flow of data needs resources such as a buffer, bandwidth, CPU time, and so on. The quality of service is improved if these resources are reserved beforehand. We discuss in

Section 25.11 one QoS model called Integrated Services, which depends heavily on resource reservation to improve the quality of service.

### Admission Control

Admission control refers to the mechanism used by a router, or a switch, to accept or reject a flow based on predefined parameters called flow specifications. Before a router accepts a flow for processing, it checks the flow specifications to see if its capacity (in terms of bandwidth, buffer size, CPU speed, etc.) and its previous commitments to other flows can handle the new flow.

## 25.11 INTEGRATED SERVICES

Based on the topics in Sections 25.10 two models have been designed to provide quality of service in the Internet: Integrated Services and Differentiated Services. Both models emphasize the use of quality of service at the network layer (IP), although the model can also be used in other layers such as the data link. We discuss Integrated Services in this section and Differentiated Service in Section 25.12.

As we learned in Chapter 7, IP was originally designed for *best-effort* delivery. This means that every user receives the same level of services. This type of delivery does not guarantee the minimum of a service, such as bandwidth, to applications such as real-time audio and video. If such an application accidentally gets extra bandwidth, it may be detrimental to other applications, resulting in congestion.

**Integrated Services,** sometimes called **IntServ,** is a *flow-based* QoS model, which means that a user needs to create a flow, a kind of virtual circuit, from the source to the destination and inform all routers of the resource requirement.

**Integrated Services is a *flow-based* QoS model designed for IP.**

### Signaling

The reader may remember that IP is a connectionless, datagram, packet-switching protocol. How can we implement a flow-based model over a connectionless protocol? The solution is a signaling protocol to run over IP that provides the signaling mechanism for making a reservation. This protocol is called **Resource Reservation Protocol (RSVP)** and will be discussed shortly.

### Flow Specification

When a source makes a reservation, it needs to define a flow specification. A flow specification has two parts: Rspec (resource specification) and Tspec (traffic specification). Rspec defines the resource that the flow needs to reserve (buffer, bandwidth, etc.). Tspec defines the traffic characterization of the flow.

## Admission

After a router receives the flow specification from an application, it decides to admit or deny the service. The decision is based on the previous commitments of the router and the current availability of the resource.

## Service Classes

Two classes of services have been defined for Integrated Services: guaranteed service and controlled-load service.

### *Guaranteed Service Class*

This type of service is designed for real-time traffic that needs a guaranteed minimum end-to-end delay. The end-to-end delay is the sum of the delays in the routers, the propagation delay in the media, and the setup mechanism. Only the first, the sum of the delays in the routers, can be guaranteed by the router. This type of service guarantees that the packets will arrive within a certain delivery time and are not discarded if flow traffic stays within the boundary of Tspec. We can say that guaranteed services are quantitative services, in which the amount of end-to-end delay and the data rate must be defined by the application.

### *Controlled-Load Service Class*

This type of service is designed for applications that can accept some delays, but are sensitive to an overloaded network and to the danger of losing packets. Good examples of these types of applications are file transfer, e-mail, and Internet access. The controlled-load service is a qualitative type of service in that the application requests the possibility of low-loss or no-loss packets.

## RSVP

In the Integrated Services model, an application program needs resource reservation. As we learned in the discussion of the IntServ model, the resource reservation is for a *flow.* This means that if we want to use IntServ at the IP level, we need to create a flow, a kind of virtual-circuit network, out of the IP, which was originally designed as a datagram packet-switched network. A virtual-circuit network needs a signaling system to set up the virtual circuit before data traffic can start. The Resource Reservation Protocol (RSVP) is a signaling protocol to help IP create a flow and consequently make a resource reservation. Before discussing RSVP, we need to mention that it is an independent protocol separate from the Integrated Services model. It may be used in other models in the future.

### *Multicast Trees*

RSVP is different from some other signaling systems we have seen before in that it is a signaling system designed for multicasting. However, RSVP can be also used for unicasting because unicasting is just a special case of multicasting with only one member in the multicast group. The reason for this design is to enable RSVP to provide resource reservations for all kinds of traffic including multimedia which often uses multicasting.

### *Receiver-Based Reservation*

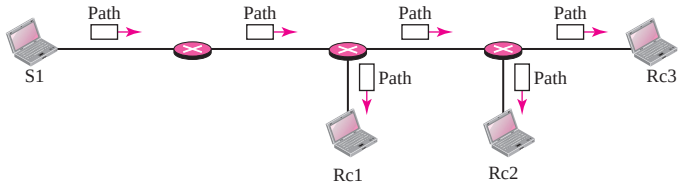
In RSVP, the receivers, not the sender, make the reservation. This strategy matches the other multicasting protocols. For example, in multicast routing protocols, the receivers, not the sender, make a decision to join or leave a multicast group.

### *RSVP Messages*

RSVP has several types of messages. However, for our purposes, we discuss only two of them: **Path** and **Resv.**

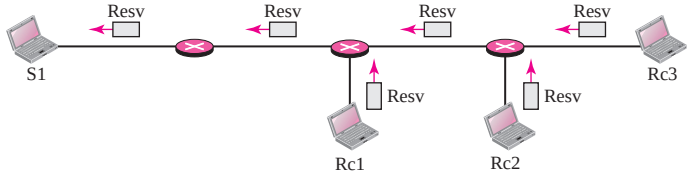
**Path Messages** Recall that the receivers in a flow make the reservation in RSVP. However, the receivers do not know the path traveled by packets before the reservation is made. The path is needed for the reservation. To solve the problem, RSVP uses *Path* messages. A Path message travels from the sender and reaches all receivers in the multicast path. On the way, a Path message stores the necessary information for the receivers. A Path message is sent in a multicast environment; a new message is created when the path diverges. Figure 25.35 shows path messages.

**Figure 25.35** *Path messages*




**Resv Messages** After a receiver has received a Path message, it sends a *Resv* message. The Resv message travels toward the sender (upstream) and makes a resource reservation on the routers that support RSVP. If a router does not support RSVP on the path, it routes the packet based on the best-effort delivery methods we discussed before. Figure 25.36 shows the Resv messages.

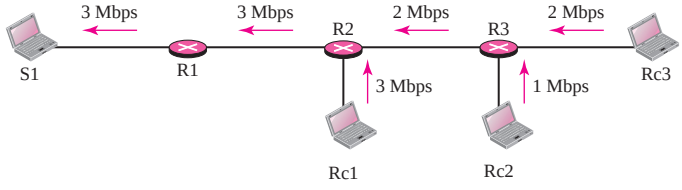
**Figure 25.36** *Resv messages*



### *Reservation Merging*

In RSVP, the resources are not reserved for each receiver in a flow; the reservation is merged. In Figure 25.37, Rc3 requests a 2-Mbps bandwidth while Rc2 requests a 1-Mbps bandwidth. Router R3, which needs to make a bandwidth reservation, merges the two requests. The reservation is made for 2 Mbps, the larger of the two, because a 2-Mbps input reservation can handle both requests. The same situation is true for R2. The reader may ask why Rc2 and Rc3, both belonging to one single flow, request different amounts of bandwidth. The answer is that, in a multimedia environment, different receivers may handle different grades of quality. For example, Rc2 may be able to receive video only at 1 Mbps (lower quality), while Rc3 may be able to receive video at 2 Mbps (higher quality).

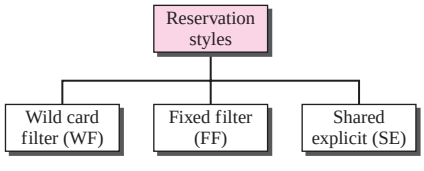
**Figure 25.37** *Reservation merging*




### *Reservation Styles*

When there is more than one flow, the router needs to make a reservation to accommodate all of them. RSVP defines three types of reservation styles, as shown in Figure 25.38.

**Figure 25.38** *Reservation styles*




**Wild Card Filter Style** In this style, the router creates a single reservation for all senders. The reservation is based on the largest request. This type of style is used when the flows from different senders do not occur at the same time.

**Fixed Filter Style** In this style, the router creates a distinct reservation for each flow. This means that if there are *n* flows, *n* different reservations are made. This type of style is used when there is a high probability that flows from different senders will occur at the same time.

**Shared Explicit Style** In this style, the router creates a single reservation that can be shared by a set of flows.

#### *Soft State*

The reservation information (state) stored in every node for a flow needs to be refreshed periodically. This is referred to as a *soft state* as compared to the *hard state* used in other virtual-circuit protocols such as ATM or Frame Relay, where the information about the flow is maintained until it is erased. The default interval for refreshing is currently 30 s.

### Problems with Integrated Services

There are at least two problems with Integrated Services that may prevent its full implementation in the Internet: scalability and service-type limitation.

#### *Scalability*

The Integrated Services model requires that each router keep information for each flow. As the Internet is growing every day, this is a serious problem.

#### *Service-Type Limitation*

The Integrated Services model provides only two types of services, guaranteed and control-load. Those opposing this model argue that applications may need more than these two types of services.

## 25.12 DIFFERENTIATED SERVICES

**Differentiated Services (DS** or **Diffserv)** was introduced by the IETF (Internet Engineering Task Force) to handle the shortcomings of Integrated Services. Two fundamental changes were made:

1. The main processing was moved from the core of the network to the edge of the network. This solves the scalability problem. The routers do not have to store information about flows. The applications, or hosts, define the type of service they need each time they send a packet.
2. The per-flow service is changed to per-class service. The router routes the packet based on the class of service defined in the packet, not the flow. This solves the service-type limitation problem. We can define different types of classes based on the needs of applications.

**Differentiated Services is a class-based QoS model designed for IP.**

### DS Field

In Diffserv, each packet contains a field called the DS field. The value of this field is set at the boundary of the network by the host or the first router designated as the boundary

router. IETF proposes to replace the existing TOS (type of service) field in IPv4 or the class field in IPv6 by the DS field, as shown in Figure 25.39.

**Figure 25.39** *DS field*


The DS field contains two subfields: DSCP and CU. The DSCP (Differentiated Services Code Point) is a 6-bit subfield that defines the **per-hop behavior (PHB).** The 2-bit CU (currently unused) subfield is not currently used.

The Diffserv capable node (router) uses the DSCP 6 bits as an index to a table defining the packet-handling mechanism for the current packet being processed.

### *Per-Hop Behavior*

The Diffserv model defines per-hop behaviors (PHBs) for each node that receives a packet. So far three PHBs are defined: DE PHB, EF PHB, and AF PHB.

**DE PHB** The DE PHB (default PHB) is the same as best-effort delivery, which is compatible with TOS.

**EF PHB** The EF PHB (expedited forwarding PHB) provides the following services:

- Low loss
- Low latency
- Ensured bandwidth

This is the same as having a virtual connection between the source and destination.

**AF PHB** The AF PHB (assured forwarding PHB) delivers the packet with a high assurance as long as the class traffic does not exceed the traffic profile of the node. The users of the network need to be aware that some packets may be discarded.

### *Traffic Conditioner*

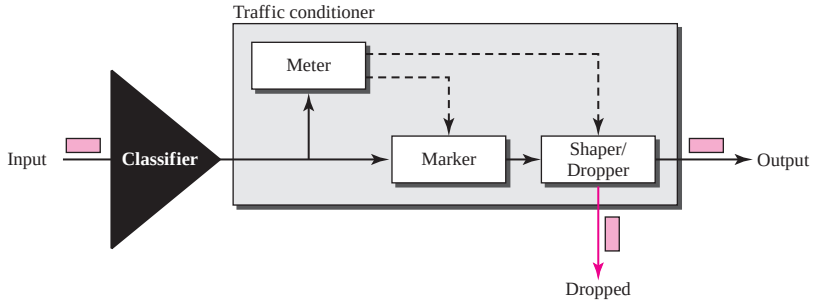
To implement Diffserv, the DS node uses traffic conditioners such as meters, markers, shapers, and droppers, as shown in Figure 25.40.

**Meters** The meter checks to see if the incoming flow matches the negotiated traffic profile. The meter also sends this result to other components. The meter can use several tools such as a token bucket to check the profile.

**Marker** A marker can remark a packet that is using best-effort delivery (DSCP: 000000) or down-mark a packet based on information received from the meter. Down-marking (lowering the class of the flow) occurs if the flow does not match the profile. A marker does not up-mark a packet (promote the class).

**Shaper** A shaper uses the information received from the meter to reshape the traffic if it is not compliant with the negotiated profile.

**Figure 25.40** *Traffic conditioner*




**Dropper** A dropper, which works as a shaper with no buffer, discards packets if the flow severely violates the negotiated profile.

## 25.13 RECOMMENDED READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give some coverage of multimedia. [Com 06] discusses voice and audio over IP. [Kur & Ros 08] and [Tan 03] have a lengthy discussion on multimedia. [Gar & Vid 04] also give a more detailed discussion about audio and video compression.

### RFCs

Several RFCs show different update on topics discussed in this chapter including RFC 2198, RFC 2250, RFC 2326, RFC 2475, RFC 3246, RFC 3550, and RFC 3551.

## 25.14 KEY TERMS

bidirectional frame (B-frame)
compression
delay
Differentiated Services (DS or Diffserv)
discrete cosine transform (DCT)
first-in, first-out (FIFO) queuing
frequency masking
gatekeeper
gateway
H.323
Integrated Services (IntServ)
interactive audio/video
intracoded frame (I-frame)
jitter
Joint Photographic Experts Group (JPEG)
leaky bucket
media player
media server

metafile
mixer
Moving Picture Experts Group (MPEG)
multicasting
on-demand audio/video
open-loop congestion control
Path message
perceptual encoding
per-hop behavior (PHB)
pixel
playback buffer
predicted frame (P-frame)
predictive encoding
priority queuing
quality of service (QoS)
quantization
Real-Time Streaming Protocol (RTSP)
Real-Time Transport Control Protocol (RTCP)
Real-Time Transport Protocol (RTP)
registrar server
reliability
Resource Reservation Protocol (RSVP)
Resv message
Session Initiation Protocol (SIP)
spatial compression
streaming live audio/video
streaming stored audio/video
temporal compression
temporal masking
timestamp
token bucket
traffic shaping
translation
voice over IP
weighted fair queuing

## 25.15 SUMMARY

- Audio/video files can be downloaded for future use (streaming stored audio/video) or broadcast to clients over the Internet (streaming live audio/video). The Internet can also be used for live audio/video interaction. Audio and video need to be digitized before being sent over the Internet.
- Audio files are compressed through predictive encoding or perceptual encoding. Joint Photographic Experts Group (JPEG) is a method to compress pictures and graphics. Moving Pictures Experts Group (MPEG) is a method to compress video.
- We can use a Web server, or a Web server with a metafile, or a media server, or a media server and RTSP to download a streaming audio/video file.
- Real-time data on a packet-switched network require the preservation of the time relationship between packets of a session. Gaps between consecutive packets at the receiver cause a phenomenon called jitter. Jitter can be controlled through the use of timestamps and a judicious choice of the playback time.
- Real-time multimedia traffic requires both UDP and Real-Time Transport Protocol (RTP). RTP handles timestamping, sequencing, and mixing. Real-Time Transport Control Protocol (RTCP) provides flow control, quality of data control, and feedback to the sources.
- Voice over IP is a real-time interactive audio/video application. The Session Initiation Protocol (SIP) is an application layer protocol that establishes, manages, and terminates multimedia sessions. H.323 is an ITU standard that allows a telephone connected to a public telephone network to talk to a computer connected to the Internet.

- A flow can be characterized by its reliability, delay, jitter, and bandwidth. Scheduling, traffic shaping, resource reservation, and admission control are techniques to improve quality of service (QoS).
- Integrated Services is a flow-based QoS model designed for IP. The Resource Reservation Protocol (RSVP) is a signaling protocol that helps IP create a flow and makes a resource reservation. Differential Services is a class-based QoS model designed for IP.

## 25.16 PRACTICE SET

### Exercises

1. In Figure 25.17 what is the amount of data in the playback buffer at each of the following times?
   a. 00:00:17
   b. 00:00:20
   c. 00:00:25
   d. 00:00:30
2. Compare and contrast TCP with RTP. Are both doing the same thing?
3. Can we say UDP plus RTP is the same as TCP?
4. Why does RTP need the service of another protocol, RTCP, but TCP does not?
5. In Figure 25.12, can the Web server and media server run on different machines?
6. We discuss the use of SIP in this chapter for audio. Is there any drawback to prevent using it for video?
7. Do you think H.323 is actually the same as SIP? What are the differences? Make a comparison between the two.
8. Can H.323 also be used for video?
9. In a leaky bucket used to control liquid flow, how many gallons of liquid are left in the bucket if the output rate is 5 gal/min, there is an input burst of 100 gal/min for 12 s, and there is no input for 48 s?
10. An output interface in a switch is designed using the leaky bucket algorithm to send 8,000 bytes/s (tick). If the following frames are received in sequence, show the frames that are sent during each second.
    a. Frames 1, 2, 3, 4: 4,000 bytes each
    b. Frames 5, 6, 7: 3,200 bytes each
    c. Frames 8, 9: 400 bytes each
    d. Frames 10, 11, 12: 2,000 bytes each

# PART 5

# Next Generation

CHAPTER

# 26

# *IPv6 Addressing*

In the first chapter of Part 5 of the book, we introduce addressing in IPv6. The address depletion of IPv4 protocol was one of the major reason for developing IPv6 protocol. As we see in this chapter, the structure of IPv6 addresses has some fundamental differences with the structure of IPv4 addresses. Depletion of address is definitely out of the question.

## OBJECTIVES

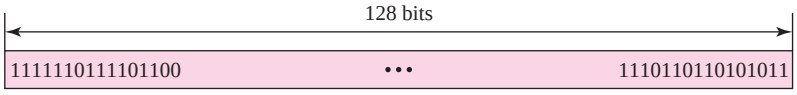
*The chapter has several objectives:*

- To introduce the IPv6 addressing scheme and different notations used to represent an address in this version.
- To explain the three types of addressing used in IPv6: unicast, anycast, and multicast.
- To show the address space in this version and how it is divided into several blocks.
- To discuss some reserved blocks in the address space and their applications.
- To define the global unicast address block and how it is used for unicast communication.
- To discuss how three levels of hierarchy in addressing are used in IPv6 deploying the global unicast block.
- To discuss autoconfiguration and renumbering of IPv6 addresses.

## 26.1 INTRODUCTION

An IPv6 address is 128 bits or 16 bytes (octet) long as shown in Figure 26.1. The address length in IPv6 is four times of the length address in IPv4.

**Figure 26.1** *IPv6 address*




### Notations

A computer normally stores the address in binary, but is clear that 128 bits cannot easily be handled by humans. Several notations have been proposed to represent IPv6 addresses when they are handled by humans:

#### *Dotted-Decimal Notation*

To be compatible with IPv4 addresses, we are tempted to use dotted-decimal notation as shown for IPv4 addresses in Chapter 5. Although this notation is convenient for 4-byte IPv4 addresses, it seems too long for 16-byte IPv6 addresses as shown below:

```
221.14.65.11.105.45.170.34.12.234.18.0.14.0.115.255
```

This notation is rarely used except partially as we see shortly.

#### *Colon Hexadecimal Notation*

To make addresses more readable, IPv6 specifies **colon hexadecimal notation** (or *colon hex* for short). In this notation, 128 bits are divided into eight sections, each 2 bytes in length. Two bytes in hexadecimal notation require four hexadecimal digits. Therefore, the address consists of 32 hexadecimal digits, with every four digits separated by a colon. Figure 26.2 shows an IPv6 address in colon hexadecimal notation.

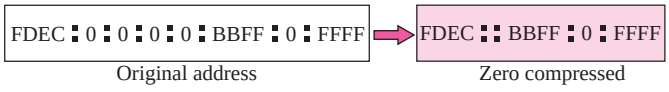
**Figure 26.2** *Colon hexadecimal notation*

FDEC : BA98 : 7654 : 3210 : ADBF : BBFF : 2922 : FFFF

Although the IP address, even in hexadecimal format, is very long, many of the digits are zeros. In this case, we can abbreviate the address. The leading zeros of a section can be omitted. Using this form of abbreviation, 0074 can be written as 74, 000F as F, and 0000 as 0. Note that 3210 cannot be abbreviated.

Further abbreviation, often called **zero compression,** can be applied to colon hex notation if there are consecutive sections consisting of zeros only. We can remove all the zeros altogether and replace them with a double semicolon. Figure 26.3 shows the concept.

**Figure 26.3** *Zero compression*




Note that this type of abbreviation is allowed only once per address. If there are two runs of zero sections, only one of them can be compressed.

### *Mixed Representation*

Sometimes we see a mixed representation of an IPv6 address: colon hex and dotted-decimal notation. This is appropriate during the transition period in which an IPv4 address is embedded in an IPv6 address (as the rightmost 32 bits). We can use the colon hex notation for the leftmost six section and four bytes dotted-decimal notation instead of the rightmost two sections as shown below:

**FDEC:14AB:2311:BBFE:AAAA:BBBB:130.24.24.18**

However, this happens when all or most of the rightmost sections of the IPv6 address are 0s. For example, the following is a legitimate address in IPv6, in which the zero compression shows that all 96 leftmost bits of the address are all zeros:

**::130.24.24.18**

### *CIDR Notation*

As we see shortly, IPv6 uses hierarchical addressing. For this reason, IPv6 allows classless addressing and CIDR notation. For example, Figure 26.4 shows how we can define a prefix of 60 bits using CIDR. We will later show how an IPv6 is divided into a prefix and a suffix.

**Figure 26.4** *CIDR address*

### Example 26.1

Show the unabbreviated colon hex notation for the following IPv6 addresses:

**a.** An address with 64 0s followed by 64 1s.
**b.** An address with 128 0s.
**c.** An address with 128 1s.
**d.** An address with 128 alternative 1s and 0s.

**Solution**

**a.** `0000:0000:0000:0000:FFFF:FFFF:FFFF:FFFF`
**b.** `0000:0000:0000:0000:0000:0000:0000:0000`
**c.** `FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF`
**d.** `AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA`

### Example 26.2

The following shows the zero contraction version of addresses in Example 26.1 (part c and d cannot be abbreviated)

**a.** `::FFFF:FFFF:FFFF:FFFF`
**b.** `::`
**c.** `FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF`
**d.** `AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA:AAAA`

### Example 26.3

Show abbreviations for the following addresses:

**a.** `0000:0000:FFFF:0000:0000:0000:0000:0000`
**b.** `1234:2346:0000:0000:0000:0000:0000:1111`
**c.** `0000:0001:0000:0000:0000:0000:1200:1000`
**d.** `0000:0000:0000:0000:0000:FFFF:24.123.12.6`

**Solution**

**a.** `0:0:FFFF::`
**b.** `1234:2346::1111`
**c.** `0:1::1200:1000`
**d.** `::FFFF:24.123.12.6`

### Example 26.4

Decompress the following addresses and show the complete unabbreviated IPv6 address:

**a.** `1111::2222`
**b.** `::`
**c.** `0:1::`
**d.** `AAAA:A:AA::1234`

**Solution**

a. **1111:0000:0000:0000:0000:0000:0000:2222**
b. **0000:0000:0000:0000:0000:0000:0000:0000**
c. **0000:0001:0000:0000:0000:0000:0000:0000**
d. **AAAA:000A:00AA:0000:0000:0000:0000:1234**

## Address Space

The address space of IPv6 contains $2^{128}$ addresses as shown below. This address space is $2^{96}$ times of the IPv4 address—definitely no address depletion.

340,282,366,920,938,463,374,607,431,768,211,456

### Example 26.5

To give some idea about the number of addresses, let us assume that the number of people on the planet earth is soon to be $2^{34}$ (more than 16 billion). Each person can have $2^{94}$ addresses to use.

### Example 26.6

If we assign $2^{60}$ addresses to the users each year (almost one billion each second), it takes $2^{68}$ years to deplete addresses.

### Example 26.7

If we can build a high-rise building over the land and sea to accommodate $2^{68}$ computers in each square meter of the earth, still there are enough addresses to connect all computers to the Internet (the planet earth is approximately $2^{60}$ square meters).

## Three Address Types

In IPv6, a destination address can belong to one of three categories: unicast, anycast, and multicast.

### *Unicast Address*

A unicast address defines a single interface (computer or router). The packet sent to a unicast address will be routed to the intended recipient. As we see shortly, IPv6 has designated a large block from which unicast addresses can be assigned to interfaces.

### *Anycast Address*

An anycast address defines a group of computers that all share a single address. A packet with an anycast address is delivered to only one member of the group, the most reachable one. An anycast communication is used, for example, when there are several servers that can respond to an inquiry. The request is sent to the one that is most reachable. The hardware and software generate only one copy of the request; the copy reaches only one of the servers. IPv6 does not designate a block for anycasting; the addresses are assigned from the unicast block.

### *Multicast Address*

A multicast address also defines a group of computers. However, there is a difference between anycasting and multicasting. In multicasting, each member of the group

receives a copy. As we will see shortly, IPv6 has designated a block for multicasting from which the same address is assigned to the members of the group.

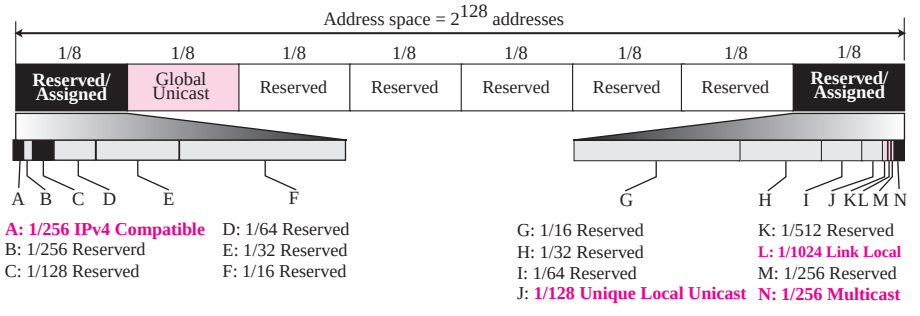
### Broadcasting and Multicasting

It is interesting that IPv6 does not define broadcasting, even in a limited version, as IPv4 does. In Chapter 5, we discussed that some addresses in a block can be used for limited broadcasting. As we will see, IPv6 considers broadcasting as a special case of multicasting.

## 26.2 ADDRESS SPACE ALLOCATION

Like the address space of IPv4, the address space of IPv6 is divided into several blocks of varying size and each block is allocated for special purpose. Most of the blocks are still unassigned and have been left aside for future use. To better understand the allocation and the location of each block in address space, we first divide the whole address space into eight equal ranges. This division does not show the block allocation, but we believe it shows where each actual block is located (Figure 26.5).

**Figure 26.5** *Address space allocation*




Each section is one-eighth of the whole address space ($2^{125}$ addresses). The first section contains six variable-size blocks; three of these blocks are reserved and three unassigned. The second section is considered one single block and is used for global unicast addresses, which we discuss later in the chapter. The next five sections are unassigned addresses. The last section is divided into eight blocks. Some of these blocks are still unassigned and some are reserved for special purposes. The figure shows that more than five-eighths of the address space is still unassigned. Only one-eighth of the address space is used for unicast communication between the users.

Table 26.1 shows the prefix for each type of address. The third column shows the fraction of each type of address relative to the whole address space. The leftmost column is not part of the standard; it shows only the section described in Figure 26.5.

**Table 26.1** *Prefixes for IPv6 Addresses*

| | *Block Prefix* | *CIDR* | *Block Assignment* | *Fraction* |
|---|---|---|---|---|
| 1 | 0000 0000 | 0000::/8 | Reserved (IPv4 compatible) | 1/256 |
| | 0000 0001 | 0100::/8 | Reserved | 1/256 |
| | 0000 001 | 0200::/7 | Reserved | 1/128 |
| | 0000 01 | 0400::/6 | Reserved | 1/64 |
| | 0000 1 | 0800::/5 | Reserved | 1/32 |
| | 0001 | 1000::/4 | Reserved | 1/16 |
| **2** | **001** | **2000::/3** | **Global unicast** | **1/8** |
| 3 | 010 | 4000::/3 | Reserved | 1/8 |
| 4 | 011 | 6000::/3 | Reserved | 1/8 |
| 5 | 100 | 8000::/3 | Reserved | 1/8 |
| 6 | 101 | A000::/3 | Reserved | 1/8 |
| 7 | 110 | C000::/3 | Reserved | 1/8 |
| 8 | 1110 | E000::/4 | Reserved | 1/16 |
| | 1111 0 | F000::/5 | Reserved | 1/32 |
| | 1111 10 | F800::/6 | Reserved | 1/64 |
| | 1111 110 | FC00::/7 | Unique local unicast | 1/128 |
| | 1111 1110 0 | FE00::/9 | Reserved | 1/512 |
| | 1111 1110 10 | FE80::/10 | Link local addresses | 1/1024 |
| | 1111 1110 11 | FEC0::/10 | Reserved | 1/1024 |
| | 1111 1111 | FF00::/8 | Multicast addresses | 1/256 |

### Example 26.8

Figure 26.5 shows that only a portion of the address space can be used for global unicast communication. How many addresses are in this block?

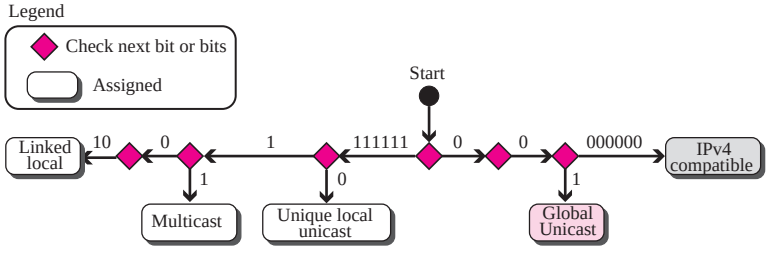
**Solution**

This block occupies only one-eighth of the address spaces. To find the number of addresses, we can divide the total address space by 8 or $2^3$. The result is $(2^{128})/(2^3) = 2^{125}$—a huge block.

### *Algorithm*

To show that the prefixes in Table 26.1 unambiguously find the block to which an IPv6 belongs to, we have created the diagram in Figure 26.6. The algorithm can be used to write a program to find the block when an address is given. The algorithm has to check only a maximum of 10 bits to find the block of the address. Note that the reserved

**Figure 26.6** *Algorithm for finding the allocated blocks*



blocks (with the exception of IPv4-compatible addresses) are not shown to make the diagram simpler.

## Assigned and Reserved Blocks

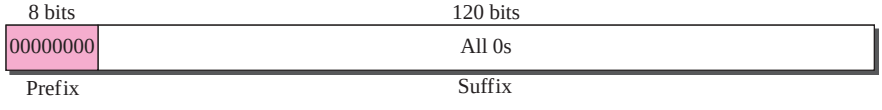
In this section, we discuss the characteristics and purposes of assigned and reserved blocks starting with the first row of Table 26.1.

### *IPv4 Compatible Addresses*

Addresses that use the prefix (00000000) are reserved, but part of it is used to define some IPv4 compatible addresses. This block occupies 1/256 of the total address space, which means that there are $2^{120}$ addresses in this block. In CIDR notation, this block can be defined as 0000::**/8**. This block is further divided into several subblocks that are discussed later.

**Unspecified Address** The unspecified address is a subblock containing only one single address, which is defined by letting all suffix bits to 0s. In other words, the entire address consists of zeros. The unspecified address is used during bootstrap when a host does not know its own address and wants to send an inquiry to find it. Since any IPv6 packet needs a source address, the host uses this address for this purpose. Note that the unspecified address cannot be used as a destination address. The CIDR notation for this one-address subblock is ::/**128**. The unspecified address format is shown in Figure 26.7.

**Figure 26.7** *Unspecified address*




**The unspecified address in IPv6 is ::/128.**
**It should never be used as a destination address.**

### Example 26.9

Comparing the unspecified address in IPv4 to the unspecified addresses in IPv6.

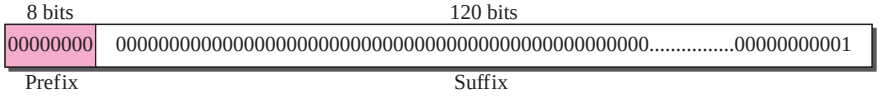
**Solution**

In both architectures, an unspecified address is an all-zero address. In IPv4 this address is part of class A address; in IPv6 this address is part of the reserved block.

**Loopback Address** This subblock also consists of one single address. We discussed loopback addresses in Chapter 5. This is an address used by a host to test itself without going into the network. In this case, a message is created in the application layer, sent to the transport layer, and passed to the network layer. However, instead of going to the physical network, it returns to the transport layer and then passes to the application layer.

This is very useful for testing the functions of software packages in these layers before even connecting the computer to the network. The loopback address as shown in Figure 26.8 consists of the prefix 00000000 followed by 119 0s and one 1. The CIDR notation for this one-address single block is ::1**/128**.

**Figure 26.8** *Loopback address*




**The loopback address in IPv6 is ::1/128.**
**It should never be used as a destination address.**

### Example 26.10

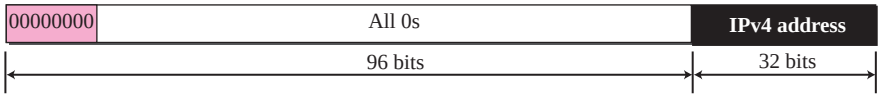
Compare the loop addresses in IPv4 to the loopback address in IPv6.

**Solution**

There are two differences in this case. In classful addressing, a whole block is allocated for loopback addresses; in IPv6 only one address is allocated as the loopback address. In addition, the loopback block in classful addressing is part of the class A block. In IPv6, it is only one single address in the reserved block.

**Embedded IPv4 Addresses** As we will see in Chapter 27, during the transition from IPv4 to IPv6, hosts can use their IPv4 addresses embedded in IPv6 addresses. Two formats have been designed for this purpose: compatible and mapped. A **compatible address** is an address of 96 bits of zero followed by 32 bits of IPv4 address. It is used when a computer using IPv6 wants to send a message to another computer using IPv6. However, suppose the packet passes through a region where the networks are still using IPv4. The sender then must use the IPv4-compatible address to facilitate the passage of the packet through the IPv4 region. For example, the IPv4 address 2.13.17.14 (in dotted decimal format) becomes 0::2.13.17.14 (in mixed format). The IPv4 address is prepended with 96 zeros to create a 128-bit IPv6 address (see Figure 26.9). This subblock is a reservation that can contain up to $2^{32}$ addresses. The CIDR notation for this subblock is ::**/96**. We will discuss more about this address in Chapter 27.

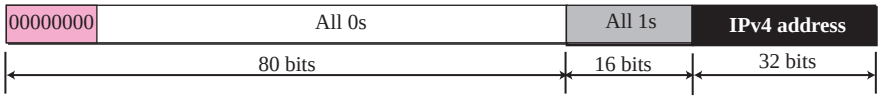
**Figure 26.9** *Compatible address*




A **mapped address** comprises 80 bits of zero, followed by 16 bits of one, followed by the 32-bit IPv4 address. It is used when a computer that has migrated to IPv6

wants to send a packet to a computer still using IPv4. The packet travels mostly through IPv6 networks but is finally delivered to a host that uses IPv4. For example, the IPv4 address 2.13.17.14 (in dotted decimal format) becomes 0::FFFF:2.13.17.14 (in hexadecimal colon format). The IPv4 address is prepended with 16 ones and 80 zeros to create a 128-bit IPv6 address (see Section 27.3 on Transition Strategies). Figure 26.10 shows a mapped address.

**Figure 26.10** *Mapped address*




A very interesting point about mapped and compatible addresses is that they are designed such that, when calculating the checksum, one can use either the embedded address or the total address because extra 0s or 1s in multiples of 16 do not have any effect in checksum calculation. This is important for UDP and TCP, which use a pseudoheader to calculate the checksum because the checksum calculation is not affected if the address of the packet is changed from IPv6 to IPv4 by a router.

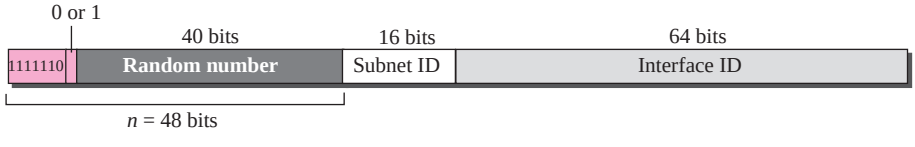
### *Global Unicast Block*

This is the main block used for unicast communication between hosts in the Internet. We will discuss this block later in full detail to show how it will be used in the Internet to provide hierarchical addressing.

### *Unique Local Unicast Block*

We discussed private addresses in Chapter 4 for IPv4 protocol. We discussed that some blocks in the IPv4 address space were reserved for private addressing. IPv6 uses two large blocks for private addressing: one at the site level and one at the link level. We discuss the first in this section and the second in the next. (See Figure 26.11.)

**Figure 26.11** *Unique local unicast block*



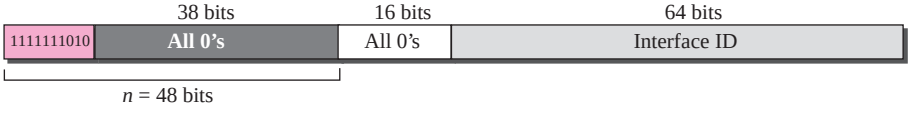

A subblock in a **unique local unicast block** can be privately created and used by a site. The packet carrying this type of address as the destination address is not expected to be routed. This type of address has the block identifier 1111 110, the next bit can be 0 or 1 to define how the address is selected (locally or by an authority). The next 40 bits are selected by the site using a randomly generated number of length 40 bits. This means that the total of 48 bits defines a subblock that looks like a global unicast

address. The 40-bit random number makes the probability of duplication of the address extremely small. Note the similarity between the format of these addresses and the global unicast address we discuss later in the chapter.

### *Link Local Block*

The second block designed for private addresses is **link local block**. A subblock in this block can be used as a private address in a network. This type of address has the block identifier 1111111010. The next 54 bits are set to zero. The last 64 bits can be changed to define the interface for each computer (see Figure 26.12). Note the similarity between the format of these addresses and the global unicast address we discuss later in the chapter.

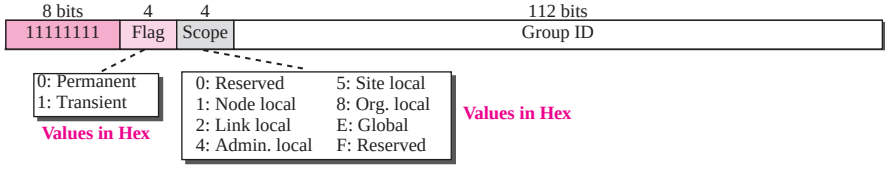
**Figure 26.12** *Link local address*




### *Multicast Block*

We discussed multicast addresses of IPv4 in Chapter 4. Multicast addresses are used to define a group of hosts instead of just one. In IPv6 a large block of addresses are assigned for multicasting. All these addresses use the prefix 11111111. The second field is a flag that defines the group address as either permanent or transient. A permanent group address is defined by the Internet authorities and can be accessed at all times. A transient group address, on the other hand, is used only temporarily. Systems engaged in a teleconference, for example, can use a transient group address. The third field defines the scope of the group address. Many different scopes have been defined, as shown in Figure 26.13.

**Figure 26.13** *Multicast address*




## 26.3 GLOBAL UNICAST ADDRESSES

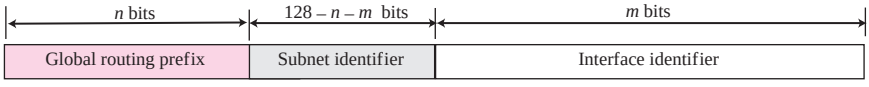
This block in the address space that is used for unicast (one-to-one) communication between two hosts in the Internet is called global unicast address block. CIDR notation for the block is 2000::/3, which means that the three leftmost bits are the same for all

addresses in this block (001). The size of this block is $2^{125}$ bits, which is more than enough for the Internet expansion in the many years to come.

## Three Levels of Hierarchy

An address in this block is divided into three parts: global routing prefix, subnet identifier, and interface identifier, as shown in Figure 26.14.

**Figure 26.14** *Global unicast address*




Recommended length of the different parts are shown in Table 26.2.

**Table 26.2** *Recommended Length of Different Parts in Unicast Addressing*

| *Block Assignment* | *Length* |
|---|---|
| Global routing prefix ($n$) | 48 bits |
| Subnet identifier ($128 - n - m$) | 16 bits |
| Interface identifier ($m$) | 64 bits |

### *Global Routing Prefix*

The first 48 bits of a global unicast address are called global routing prefix. These 48 bits are used to route the packet through the Internet to the organization site such as ISP that owns the block. Since the first three bits in this part is fixed (001), the rest of the 45 bits can defined up to $2^{45}$ sites (a private organization or an ISP). The global routers in the Internet route a packet to its destination site based on the value of $n$.

### *Subnet Identifier*

The next 16 bits defines a subnet in an organization. This means that an organization can have up to $2^{16} = 6553$ subnets, which is more than enough.

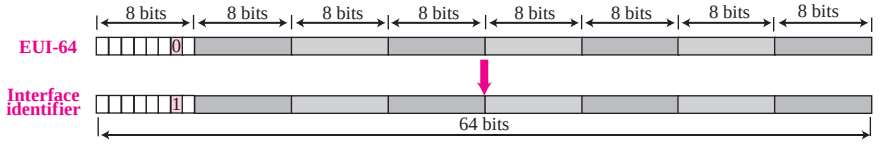
### *Interface Identifier*

The last 64 bits define the interface identifier. The interface identifier is similar to hostid in IPv4 addressing although the term interface identifier is a better choice because, as we discussed in Chapter 5, the host identifier actually defines the interface not the host. If the host is moved from one interface to another, its IP address needs to be changed.

In IPv4 addressing, there is not a specific relation between the hostid (at the IP level) and physical or MAC address (at the data link layer) because the physical address is normally much longer than the hostid. For example, using the Ethernet technology, the physical address is 48 bits while the hostid is less than 32 bits. The IPv6 addressing allows this opportunity. A physical address whose length is less than 64 bits can be embedded as the whole or part of the interface identifier, eliminating the mapping process. Two common physical addressing scheme can be considered for this purpose: the 64-bit extended unique identifier (EUI-64) defined by IEEE and the 48-bit physical address defined by Ethernet.

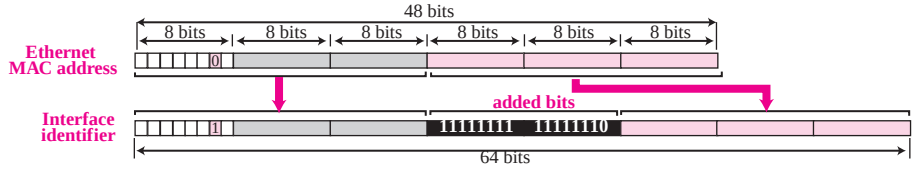
**Mapping EUI-64** To map a 64-bit physical address, the global/local bit of this format needs to be changed from 0 to 1 (local to global) to define an interface address, as shown in Figure 26.15.

**Figure 26.15** *Mapping for EUI-64*




**Mapping Ethernet MAC Address** Mapping a 48-bit Ethernet address into a 64-bit interface identifier is more involved. We need to change the local/global bit to 1 and insert an additional 16 bits. The additional 16 bits are defined as 15 ones followed by one zero, or $FFFE_{16}$. Figure 26.16 shows the mapping.

**Figure 26.16** *Mapping for Ethernet MAC*




### Example 26.11

Find the interface identifier if the physical address in the EUI is $(\mathbf{F5\text{-}A9\text{-}23\text{-}EF\text{-}07\text{-}14\text{-}7A\text{-}D2})_{16}$ using the format we defined for Ethernet addresses.

**Solution**

We only need to change the seventh bit of the first octet from 0 to 1 and change the format to colon hex notation. The result is **F7A9:23EF:0714:7AD2**.

### Example 26.12

Find the interface identifier if the Ethernet physical address is $(\mathbf{F5\text{-}A9\text{-}23\text{-}14\text{-}7A\text{-}D2})_{16}$ using the format we defined for Ethernet addresses.

**Solution**

We only need to change the seventh bit of the first octet from 0 to 1, insert two octet $FFFE_{16}$ and change the format to colon hex notation. The result is **F7A9:23FF:FE14:7AD2** in colon hex.

### Example 26.13

An organization is assigned the block 2000:1456:2474/48. What is the CIDR notation for the blocks in the first and second subnets in this organization.

**Solution**

Theoretically, the first and second subnets should use the block with subnet identifier $0001_{16}$ and $0002_{16}$. This means that the blocks are 2000:1456:2474:0000/64 and 2000:1456:2474:0001/64.

### Example 26.14

An organization is assigned the block 2000:1456:2474/48. What is the IPv6 address of an interface in the third subnet if the IEEE physical address of the computer is $(\mathbf{F5\text{-}A9\text{-}23\text{-}14\text{-}7A\text{-}D2})_{16}$.

**Solution**

The interface identifier for this interface is **F7A9:23FF:FE14:7AD2** (see Example 26.12). If we this identifier to the global prefix and the subnet identifier, we get:

```
2000:1456:2474:0003:F7A9:23FF:FE14:7AD2/128
```

## 26.4 AUTOCONFIGURATION

One of the interesting features of IPv6 addressing is the **autoconfiguration** of hosts. As we discussed in IPv4, the host and routers are originally configured manually by the network manager. However, the Dynamic Host Configuration Protocol, DHCP, can be used to allocate an IPv4 address to a host that joins the network. In IPv6, DHCP protocol can still be used to allocate an IPv6 address to a host, but a host can also configure itself.

When a host in IPv6 joins a network, it can configure itself using the following process:

1. The host first creates a link local address for itself. This is by taking the 10-bit link local prefix (1111 1110 10), adding 54 zeros, and adding the 64-bit interface identifier, which any host knows how to generate it from its interface card. The result is a 128-bit link local address.
2. The host then tests to see if this link local address is unique and not used by other hosts. Since the 64-bit interface identifier is supposed to be unique, the link local address generated is unique with a high probability. However, to be sure, the host sends a *neighbor solicitation message* (see Chapter 28) and waits for *neighbor advertisement message.* If any host in the subnet is using this link local address, the process fails and the host cannot autoconfigure itself; it needs to use other means such as DHCP protocol for this purpose.
3. If the uniqueness of the link local address is passed, the host stores this address as its link-local address (for private communication), but it still needs a global unicast address. The host then sends a *router solicitation message* (see Chapter 28) to a local router. If there is a router running on the network, the host receives a *router advertisement message* that includes the global unicast prefix and the subnet prefix that the host needs to add to its interface identifier to generate its global unicast address. If the router cannot help the host with the configuration, it informs the host in the *router advertisement message* (by setting a flag). The host then needs to use other means for configuration.

### Example 26.15

Assume a host with Ethernet address $(\mathbf{F5\text{-}A9\text{-}23\text{-}11\text{-}9B\text{-}E2})_{16}$ has joined the network. What would be its global unicast address if the global unicast prefix of the organization is 3A21:1216:2165 and the subnet identifier is A245:1232.

**Solution**

The host first creates its interface identifier as **F7A9:23FF:FE11:9BE2** using the Ethernet address read from its card. The host then creates its link-local address as

```
FE80::F7A9:23FF:FE11:9BE2
```

Assuming that this address is unique, the host sends a router solicitation message and receives the router advertisement message that announces the combination of global unicast prefix and the subnet identifier as 3A21:1216:2165:A245:1232. The host then appends its interface identifier to this prefix to find and store its global unicast address as:

```
3A21:1216:2165:A245:1232:F7A9:23FF:FE11:9BE2
```

## 26.5 RENUMBERING

To allow sites to change the service provider, **renumbering** of the address prefix (*n*) was built into IPv6 addressing. As we discussed before, each site is given a prefix by the service provider to which it is connected. If the site changes the provider, the address prefix needs to be changed. A router to which the site is connected can advertise a new prefix and let the site use the old prefix for a short time before disabling it. In other words, during the transition period, a site has two prefixes. The main problem in using the renumbering mechanism is the support of the DNS, which needs to propagate the new addressing associated with a domain name. A new protocol for DNS, called Next Generation DNS, is under study to provide support for this mechanism.

## 26.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of IPv6. We recommend [Com 06] and [Los 04].

### RFCs

Several RFCs show updates on IPv6 addressing, including RFC 2375, RFC 2526, RFC 3513, RFC 3587, RFC 3789, and RFC 4291.

## 26.7 KEY TERMS

anycast address
autoconfiguration
colon hexadecimal notation
compatible address
link local address
link local block
mapped address
renumbering
unique local unicast block
zero compression

## 26.8 SUMMARY

- IPv6, the latest version of the Internet Protocol, has a 128-bit address space. IPv6 uses hexadecimal colon notation with abbreviation methods available. There are three types of addresses: unicast, anycast, and multicast. The variable-type prefix field defines the address type or purpose.
- To make addresses more readable, IPv6 specifies colon hexadecimal notation. In this notation, 128 bits are divided into eight sections, each 2 bytes (four hexadecimal digits) in length. To abbreviate an address, the leading zeros of a section can be omitted. Zero compression, can also be applied. IPv6 also allows CIDR notation.
- In IPv6, a destination address can belong to one of the three categories: unicast, anycast, and multicast. A unicast address defines a single interface. An anycast address defines a group of computers only one copy of a packet is sent to one of the computers in the group. A multicast address also defines a group of computers; each member of the group receives a copy of the packet.
- The address space of IPv6 is divided into several blocks of varying size and each block is allocated for a special purpose. Most of the blocks are still unassigned and have been left aside for future use. Some blocks are used for reserved addresses. The most important block is the one with prefix 001, which is used for global unicast addressing (similar to class A, B, C in IPv4 addressing).
- Two interesting features of IPv6 addressing are autoconfiguration and numbering. In IPv6, a host can be autoconfigured itself in addition to using DHCP. Renumbering allows a site to change its connectivity to another provider and automatically receive a new prefix.

## 26.9 PRACTICE SET

### Exercises

1. Show the unabbreviated colon hex notation for the following IPv6 addresses:
   a. An address with 64 0s followed by 32 two-bit (01)s.
   b. An address with 64 0s followed by 32 two-bit (10)s.

**c.** An address with 64 two-bit (01)s.

**d.** An address with 32 four-bit (0111)s.

**2.** Show the zero contraction version of addresses in Exercise 1.

**3.** Show abbreviations for the following addresses:

**a.** `0000:FFFF:FFFF:0000:0000:0000:0000:0000`

**b.** `1234:2346:3456:0000:0000:0000:0000:FFFF`

**c.** `0000:0001:0000:0000:0000:FFFF:1200:1000`

**d.** `0000:0000:0000:0000:FFFF:FFFF:24.123.12.6`

**4.** Decompress the following addresses and show the complete unabbreviated IPv6 address:

**a.** `::2222`

**b.** `1111::`

**c.** `0:1:2::`

**d.** `B:A:CC::1234:A`

**5.** Show the original (unabbreviated) form of the following addresses:

**a.** 0::2

**b.** 0:23::0

**c.** 0:A::3

**d.** 123::12:23

**6.** What is the corresponding block or subblock associated with each of the following addresses based on Table 26.1:

**a.** FE80::12

**b.** FEC0::24A2

**c.** FF02::0

**d.** 0::01

**7.** What is the corresponding block or subblock associated with each of the following addresses based on Table 26.1:

**a.** 0::0

**b.** 0::FFFF:0:0

**c.** 582F:1234::2222

**d.** 4821::14:22

**8.** Find the interface identifier if the physical address of the EUI is $(F5\text{-}A9\text{-}23\text{-}AA\text{-}07\text{-}14\text{-}7A\text{-}23)_{16}$ using the format we defined for Ethernet addresses.

**9.** Find the interface identifier if the Ethernet physical address is $(F5\text{-}A9\text{-}23\text{-}12\text{-}7A\text{-}B2)_{16}$ using the format we defined for Ethernet addresses.

**10.** An organization is assigned the block 2000:1234:1423/48. What is the CIDR notation for the blocks in the first and second subnets in this organization?

**11.** An organization is assigned the block 2000:1110:1287/48. What is the IPv6 address of an interface in the third subnet if the IEEE physical address of the computer is $(F5\text{-}A9\text{-}23\text{-}14\text{-}7A\text{-}D2)_{16}$.

**12.** Using the CIDR notation, show the IPv6 address compatible to the IPv4 address 129.6.12.34.

13. Using the CIDR notation, show the IPv6 address mapped to the IPv4 address 129.6.12.34.
14. Using the CIDR notation, show the IPv6 loopback address.
15. Using the CIDR notation, show the link local address in which the node identifier is 0::123/48.
16. Using the CIDR notation, show the site local address in which the node identifier is 0::123/48.

CHAPTER

# 27

# IPv6 Protocol

In Chapter 26, we discussed IPv6 addressing space and showed how this space can remove the depletion problems we have encountered in IPv4. The address change requires that the format of the IP datagram to be changed to accommodate 128-bit addresses. As we will see in this chapter, the new IPv6 protocol has also responded to some problems encountered during the operation of IPv4 in the last few decades. This chapter is totally devoted to the format of IPv6 datagram. In the next chapter, we discuss how ICMPv4 has also been changed to ICMPv6 to respond to some shortcoming in this protocol.

## OBJECTIVES

*The chapter has several objectives:*

- To give the format of an IPv6 datagram composed of a base header and a payload.
- To discuss different fields used in an IPv6 datagram based header and compare them with the fields in IPv4 datagram.
- To show how the options in IPv4 header are implemented using the extension header in IPv6.
- To show how security is implemented in IPv6.
- To discuss three strategies used to handle the transition from IPv4 to IPv6: dual stack, tunneling, and header translation.

## 27.1 INTRODUCTION

In this introductory section, we discuss two topics: rationale for a new protocol and the reasons for delayed adoption.

### Rationale for Change

We can mention several reasons for the need of a new protocol, **Internet Protocol version 6 (IPv6)**. The main reason was the address depletion that we discussed in the previous chapter. Other reasons are related to the slowness of the process due to some unnecessary processing, the need for new options, support for multimedia, and the desperate need for security.

IPv6 protocol responds to the above issues using the following main changes in the protocol:

- **Larger address space.** An IPv6 address is 128 bits long. Compared with the 32-bit address of IPv4, this is a huge ($2^{96}$ times) increase in the address space.
- **Better header format.** IPv6 uses a new header format in which options are separated from the base header and inserted, when needed, between the base header and the upper-layer data. This simplifies and speeds up the routing process because most of the options do not need to be checked by routers.
- **New options.** IPv6 has new options to allow for additional functionalities.
- **Allowance for extension.** IPv6 is designed to allow the extension of the protocol if required by new technologies or applications.
- **Support for resource allocation.** In IPv6, the type-of-service field has been removed, but two new fields, *traffic class* and *flow label* have been added to enable the source to request special handling of the packet. This mechanism can be used to support traffic such as real-time audio and video.
- **Support for more security.** The encryption and authentication options in IPv6 provide confidentiality and integrity of the packet.

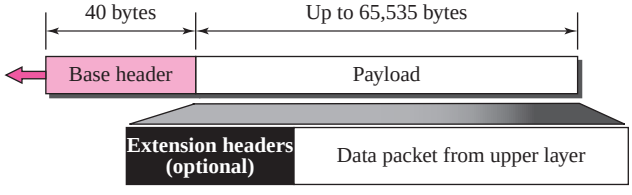
### Reason for Delay in Adoption

The adoption of IPv6 has been slow. The reason is that the original motivation for its development, depletion of IPv4 addresses, has been slowed down because of three short-term remedies: classless addressing, use of DHCP for dynamic address allocation, and NAT. However, the fast-spreading use of the Internet, and new services, such as mobile IP, IP telephony, and IP-capable mobile telephony, may require the total replacement of IPv4 with IPv6. It was predicted that all hosts in the world will be using IPv6 in 2010, but this looks unlikely at the time the book goes to press.

## 27.2 PACKET FORMAT

The IPv6 packet is shown in Figure 27.1. Each packet is composed of a mandatory base header followed by the payload. The payload consists of two parts: optional extension headers and data from an upper layer. The base header occupies 40 bytes, whereas the extension headers and data from the upper layer contain up to 65,535 bytes of information.

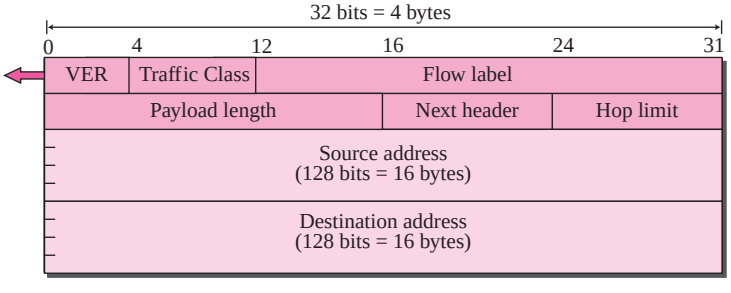
**Figure 27.1** *IPv6 datagram*




### Base Header

Figure 27.2 shows the **base header** with its eight fields.

**Figure 27.2** *Format of the base header*




These fields are as follows:

- **Version.** This 4-bit field defines the version number of the IP. For IPv6, the value is 6.
- **Traffic Class.** This 8-bit field is used to distinguish different payloads with different delivery requirements. It replaces the service class field in IPv4.
- **Flow label.** The **flow label** is a 20-bit field that is designed to provide special handling for a particular flow of data. We will discuss this field later.
- **Payload length.** The 2-byte payload length field defines the length of the IP datagram excluding the base header.

- **Next header.** The **next header** is an 8-bit field defining the header that follows the base header in the datagram. The next header is either one of the optional extension headers used by IP or the header of an encapsulated packet such as UDP or TCP. Each extension header also contains this field. Table 27.1 shows the values of next headers. Note that this field in version 4 is called the *protocol*.

**Table 27.1** *Next Header Codes*

| *Code* | *Next Header* | *Code* | *Next Header* |
|---|---|---|---|
| 0 | Hop-by-hop option | 44 | Fragmentation |
| 2 | ICMP | 50 | Encrypted security payload |
| 6 | TCP | 51 | Authentication |
| 17 | UDP | 59 | Null (No next header) |
| 43 | Source routing | 60 | Destination option |

- **Hop limit.** This 8-bit **hop limit** field serves the same purpose as the TTL field in IPv4.
- **Source address.** The source address field is a 16-byte (128-bit) Internet address that identifies the original source of the datagram.
- **Destination address.** The destination address field is a 16-byte (128-bit) Internet address that usually identifies the final destination of the datagram. However, if source routing is used, this field contains the address of the next router.

## Flow Label

The IP protocol was originally designed as a connectionless protocol. However, as we discussed in Chapters 4 and 6, the tendency is to use the IP protocol as a connection-oriented protocol. The MPLS technology described in Chapter 6 allows us to encapsulate an IPv4 packet in an MPLS header using a label field. In version 6, the *flow label* has been directly added to the format of the IPv6 datagram to allow us to use IPv6 as as connection-oriented protocol.

To a router, a flow is a sequence of packets that share the same characteristics, such as traveling the same path, using the same resources, having the same kind of security, and so on. A router that supports the handling of flow labels has a flow label table. The table has an entry for each active flow label; each entry defines the services required by the corresponding flow label. When the router receives a packet, it consults its flow label table to find the corresponding entry for the flow label value defined in the packet. It then provides the packet with the services mentioned in the entry. However, note that the flow label itself does not provide the information for the entries of the flow label table; the information is provided by other means such as the hop-by-hop options or other protocols.

In its simplest form, a flow label can be used to speed up the processing of a packet by a router. When a router receives a packet, instead of consulting the routing table and going through a routing algorithm to define the address of the next hop, it can easily look in a flow label table for the next hop.

In its more sophisticated form, a flow label can be used to support the transmission of real-time audio and video. Real-time audio or video, particularly in digital form,

requires resources such as high bandwidth, large buffers, long processing time, and so on. A process can make a reservation for these resources beforehand to guarantee that real-time data will not be delayed due to a lack of resources. The use of real-time data and the reservation of these resources require other protocols such as Real-Time Protocol (RTP) and Resource Reservation Protocol (RSVP) in addition to IPv6 (see Chapter 25).

To allow the effective use of flow labels, three rules have been defined:

1. The flow label is assigned to a packet by the source host. The label is a random number between 1 and $2^{24} - 1$. A source must not reuse a flow label for a new flow while the existing flow is still alive.
2. If a host does not support the flow label, it sets this field to zero. If a router does not support the flow label, it simply ignores it.
3. All packets belonging to the same flow have the same source, same destination, same priority, and same options.

## Comparison between IPv4 and IPv6 Headers

The following shows the comparison between IPv4 and IPv6 headers.

- The header length field is eliminated in IPv6 because the length of the header is fixed in this version.
- The service type field is eliminated in IPv6. The traffic class and flow label fields together take over the function of the service type field.
- The total length field is eliminated in IPv6 and replaced by the payload length field.
- The identification, flag, and offset fields are eliminated from the base header in IPv6. They are included in the fragmentation extension header.
- The TTL field is called hop limit in IPv6.
- The protocol field is replaced by the next header field.
- The header checksum is eliminated because the checksum is provided by upper layer protocols; it is therefore not needed at this level.
- The option fields in IPv4 are implemented as extension headers in IPv6.

## Extension Headers

The length of the base header is fixed at 40 bytes. However, to give more functionality to the IP datagram, the base header can be followed by up to six **extension headers.** Many of these headers are options in IPv4. Figure 27.3 shows the extension header format.

Six types of extension headers have been defined. These are hop-by-hop option, source routing, fragmentation, authentication, encrypted security payload, and destination option (see Figure 27.4).

### *Hop-by-Hop Option*

The **hop-by-hop option** is used when the source needs to pass information to all routers visited by the datagram. For example, perhaps routers must be informed about

**Figure 27.3** *Extension header format*

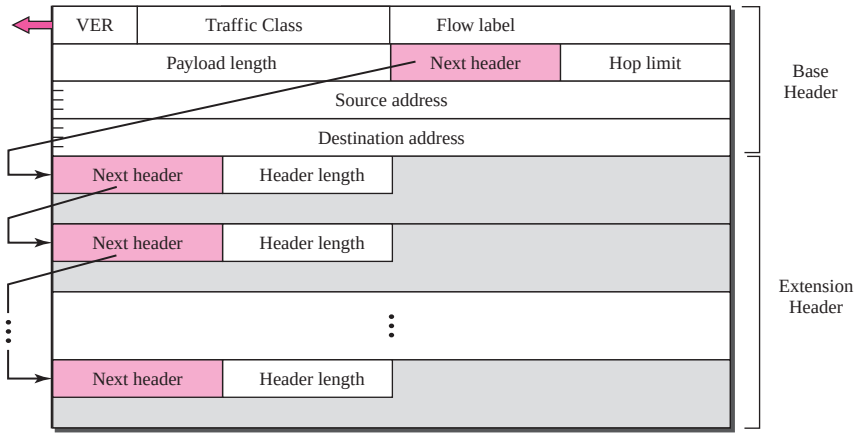
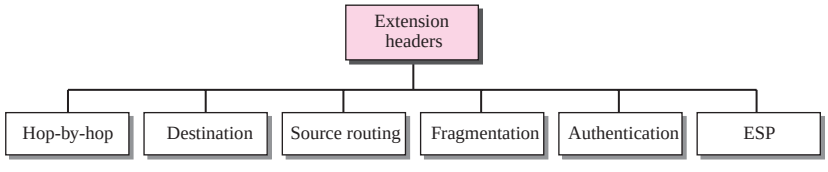
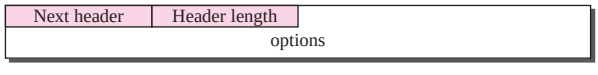



**Figure 27.4** *Extension header types*




certain management, debugging, or control functions. Or, if the length of the datagram is more than the usual 65,535 bytes, routers must have this information. Figure 27.5 shows the format of the hop-by-hop option header. The first field defines the next header in the chain of headers. The header length defines the number of bytes in the header (including the next header field). The rest of the header contains different options.

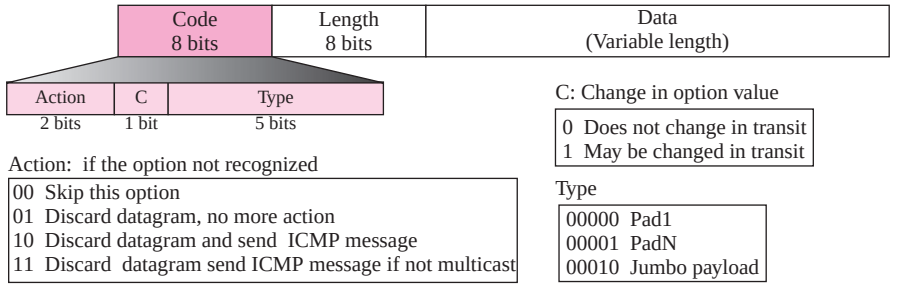
**Figure 27.5** *Hop-by-hop option header format*




So far, only three hop-by-hop options have been defined: **Pad1, PadN,** and **jumbo payload.** Figure 27.6 shows the general format of the option.

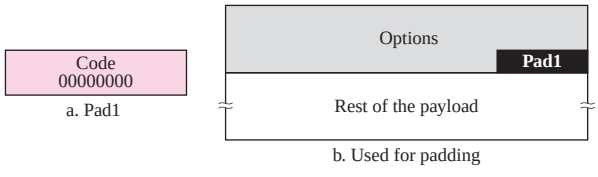
- **Pad1.** This option is 1 byte long and is designed for alignment purposes. Some options need to start at a specific bit of the 32-bit word (see the jumbo payload description to come). If an option falls short of this requirement by exactly one

**Figure 27.6** *The format of options in a hop-by-hop option header*



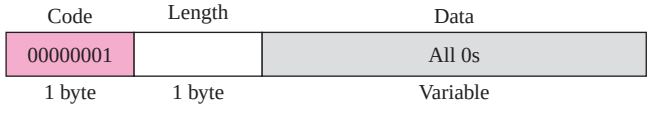

byte, Pad1 is added to make up the difference. Pad1 contains neither the option length field nor the option data field. It consists solely of the option code field with all bits set to 0 (action is 00, the change bit is 0, and type is 00000). Pad1 can be inserted anywhere in the hop-by-hop option header (see Figure 27.7).

**Figure 27.7** *Pad1*




❑ **PadN.** PadN is similar in concept to Pad1. The difference is that PadN is used when 2 or more bytes are needed for alignment. This option consists of 1 byte of option code, 1 byte of the option length, and a variable number of zero padding bytes. The value of the option code is 1 (action is 00, the change bit is 0, and type is 00001). The option length contains the number of padding bytes. See Figure 27.8.

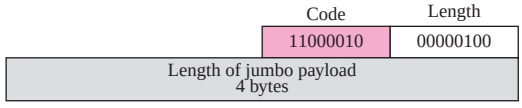
**Figure 27.8** *PadN*




❑ **Jumbo payload.** Recall that the length of the payload in the IP datagram can be a maximum of 65,535 bytes. However, if for any reason a longer payload is required, we can use the jumbo payload option to define this longer length. The jumbo payload option must always start at a multiple of 4 bytes plus 2 from the

beginning of the extension headers. The jumbo payload option starts at the $(4n + 2)$ byte, where $n$ is a small integer. See Figure 27.9.

**Figure 27.9** *Jumbo payload*



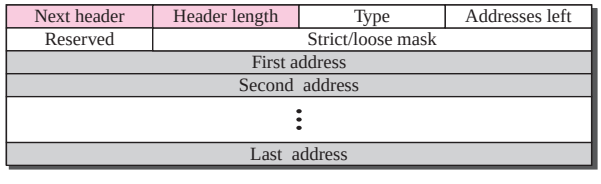

### *Destination Option*

The **destination option** is used when the source needs to pass information to the destination only. Intermediate routers are not permitted access to this information. The format of the destination option is the same as the hop-by-hop option (refer back to Figure 27.5). So far, only the Pad1 and PadN options have been defined.

### *Source Routing*

The source routing extension header combines the concepts of the strict source route and the loose source route options of IPv4. The source routing header contains a minimum of seven fields (see Figure 27.10). The first two fields, next header and header length, are identical to that of the hop-by-hop extension header. The type field defines loose or strict routing. The addresses left field indicates the number of hops still needed to reach the destination. The strict/loose mask field determines the rigidity of routing. If set to strict, routing must follow exactly as indicated by the source. If, instead, the mask is loose, other routers may be visited in addition to those in the header.

**Figure 27.10** *Source routing*

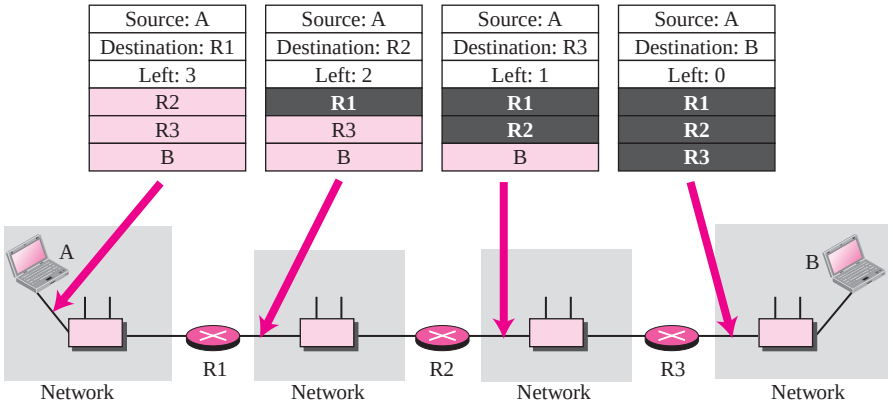



The destination address in source routing does not conform to our previous definition (the final destination of the datagram). Instead, it changes from router to router. For example, in Figure 27.11, Host A wants to send a datagram to Host B using a specific route: A to R1 to R2 to R3 to B. Notice the destination address in the base headers. It is not constant as you might expect. Instead, it changes at each router. The addresses in the extension headers also change from router to router.

### *Fragmentation*

The concept of **fragmentation** is the same as that in IPv4. However, the place where fragmentation occurs differs. In IPv4, the source or a router is required to fragment if

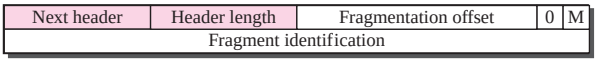
**Figure 27.11** *Source routing example*




the size of the datagram is larger than the MTU of the network over which the datagram travels. In IPv6, only the original source can fragment. A source must use a **Path MTU Discovery technique** to find the smallest MTU supported by any network on the path. The source then fragments using this knowledge.

If the source does not use a Path MTU Discovery technique, it fragments the datagram to a size of 1,280 bytes or smaller. This is the minimum size of MTU required for each network connected to the Internet. Figure 27.12 shows the format of the fragmentation extension header.

**Figure 27.12** *Fragmentation*




### *Authentication*

The **authentication** extension header has a dual purpose: it validates the message sender and ensures the integrity of data. The former is needed so the receiver can be sure that a message is from the genuine sender and not from an imposter. The latter is needed to check that the data is not altered in transition by some hacker.

The format of the authentication extension header is shown in Figure 27.13. The security parameter index field defines the algorithm used for authentication. The authentication data field contains the actual data generated by the algorithm. We will discuss authentication in Chapter 29.

Many different algorithms can be used for authentication. Figure 27.14 outlines the method for calculating the authentication data field. The sender passes a 128-bit security key, the entire IP datagram, and the 128-bit security key again to the algorithm. Those fields in the datagram with values that change during transmission (for example, hop count)

**Figure 27.13** *Authentication*

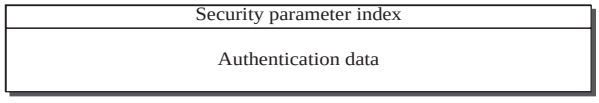
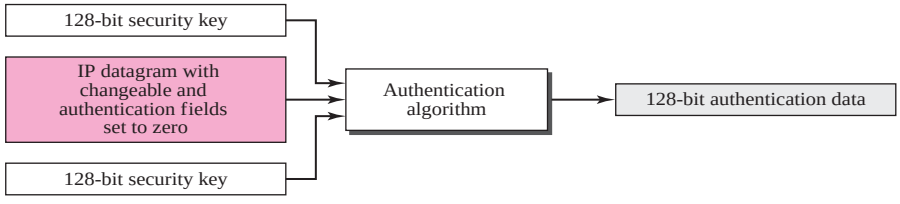



**Figure 27.14** *Calculation of authentication data*



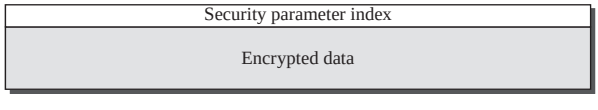

are set to zero. The datagram passed to the algorithm includes the authentication header extension, with the authentication data field set to zero. The algorithm creates authentication data which is inserted into the extension header prior to datagram transmission.

The receiver functions in a similar manner. It takes the secret key and the received datagram (again, with changeable fields set to zero) and passes them to the authentication algorithm. If the result matches that in the authentication data field, the IP datagram is authentic; otherwise, the datagram is discarded.

### *Encrypted Security Payload*

The **encrypted security payload (ESP)** is an extension that provides confidentiality and guards against eavesdropping. Figure 27.15 shows the format. The security parameter index field is a 32-bit word that defines the type of encryption/decryption used. The other field contains the encrypted data along with any extra parameters needed by the algorithm. **Encryption** can be implemented in two ways: transport mode or tunnel mode, which we discuss when we discuss IPSec in Chapter 30.

**Figure 27.15** *Encrypted security payload*




## Comparison between IPv4 and IPv6

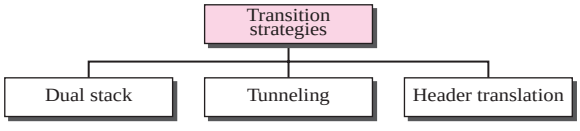
The following shows a quick comparison between the options used in IPv4 and the options used in IPv6 (as extension headers).

- The no-operation and end-of-option options in IPv4 are replaced by Pad1 and PadN options in IPv6.
- The record route option is not implemented in IPv6 because it was not used.
- The timestamp option is not implemented because it was not used.
- The source route option is called the source route extension header in IPv6.
- The fragmentation fields in the base header section of IPv4 have moved to the fragmentation extension header in IPv6.
- The authentication extension header is new in IPv6.
- The encrypted security payload extension header is new in IPv6.

## 27.3 TRANSITION FROM IPv4 TO IPv6

Because of the huge number of systems on the Internet, the transition from IPv4 to IPv6 cannot happen suddenly. It will take a considerable amount of time before every system in the Internet can move from IPv4 to IPv6. The transition must be smooth to prevent any problems between IPv4 and IPv6 systems. Three strategies have been devised by the IETF to help the transition (see Figure 27.16).

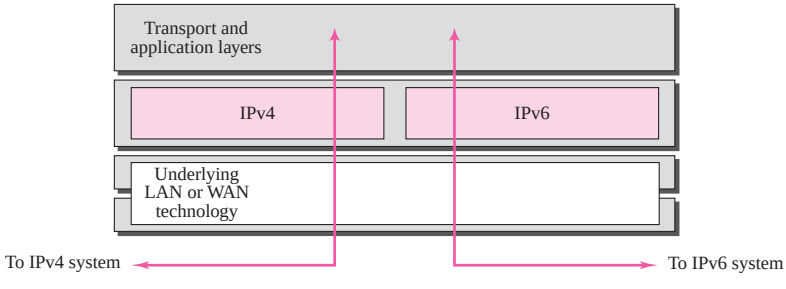
**Figure 27.16** *Three transition strategies*




### Dual Stack

It is recommended that all hosts, before migrating completely to version 6, have a **dual stack** of protocols. In other words, a station must run IPv4 and IPv6 simultaneously until all the Internet uses IPv6. See Figure 27.17 for the layout of a dual-stack configuration.

**Figure 27.17** *Dual stack*

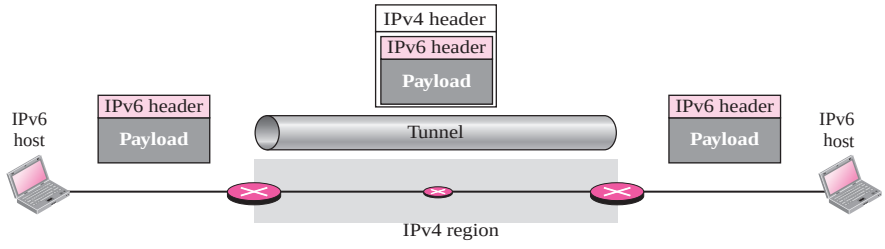


To determine which version to use when sending a packet to a destination, the source host queries the DNS. If the DNS returns an IPv4 address, the source host sends an IPv4 packet. If the DNS returns an IPv6 address, the source host sends an IPv6 packet.

## Tunneling

**Tunneling** is a strategy used when two computers using IPv6 want to communicate with each other and the packet must pass through a region that uses IPv4. To pass through this region, the packet must have an IPv4 address. So the IPv6 packet is encapsulated in an IPv4 packet when it enters the region, and it leaves its capsule when it exits the region. It seems as if the IPv6 packet goes through a tunnel at one end and emerges at the other end. To make it clear that the IPv4 packet is carrying an IPv6 packet as data, the protocol value is set to 41. Tunneling is shown in Figure 27.18.

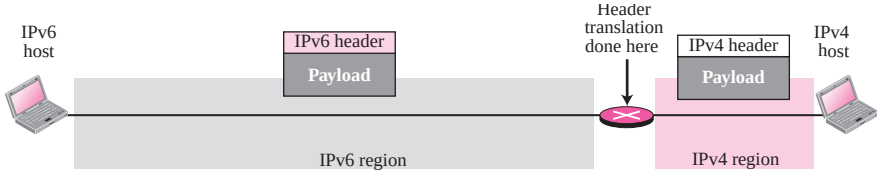
**Figure 27.18** *Tunneling strategy*




## Header Translation

**Header translation** is necessary when the majority of the Internet has moved to IPv6 but some systems still use IPv4. The sender wants to use IPv6, but the receiver does not understand IPv6. Tunneling does not work in this situation because the packet must be in the IPv4 format to be understood by the receiver. In this case, the header format must be totally changed through header translation. The header of the IPv6 packet is converted to an IPv4 header (see Figure 27.19).

**Figure 27.19** *Header translation strategy*




Header translation uses the mapped address to translate an IPv6 address to an IPv4 address. The following lists some rules used in transforming an IPv6 packet header to an IPv4 packet header.

- The IPv6 mapped address is changed to an IPv4 address by extracting the rightmost 32 bits.
- The value of the IPv6 priority field is discarded.
- The type of service field in IPv4 is set to zero.
- The checksum for IPv4 is calculated and inserted in the corresponding field.
- The IPv6 flow label is ignored.
- Compatible extension headers are converted to options and inserted in the IPv4 header. Some may have to be dropped.
- The length of IPv4 header is calculated and inserted into the corresponding field.
- The total length of the IPv4 packet is calculated and inserted in the corresponding field.

## 27.4 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of IPv6. We recommend [Com 06], [Los 04], and [Tan 03].

### RFCs

Several RFCs show updates on IPv6, including RFC 2460, RFC 2461, and RFC 2462.

## 27.5 KEY TERMS

attended bulk data traffic
authentication
background data
base header
destination option
dual stack
encrypted security payload (ESP)
encryption
extension header
flow label
fragmentation
header translation
hop limit
hop-by-hop option
Internet Protocol version 6 (IPv6)
jumbo payload
next header
Pad1
PadN
Path MTU Discovery technique
tunneling
unattended data traffic

## 27.6 SUMMARY

- An IPv6 datagram is composed of a base header and a payload. The 40-byte base header consists of the version, priority, flow label, payload length, next header, hop limit, source address, and destination address fields. The priority field is a measure of the importance of a datagram. The flow label identifies the special-handling needs of a sequence of packets.
- A payload consists of optional extension headers and data from an upper layer. Extension headers add functionality to the IPv6 datagram. The hop-by-hop option is used to pass information to all routers in the path. The source routing extension is used when the source wants to specify the transmission path. The fragmentation extension is used if the payload is a fragment of a message. The authentication extension validates the sender of the message and protects the data from hackers. The encrypted security payload extension provides confidentiality between sender and receiver. The destination extension passes information from the source to the destination exclusively.
- Three strategies used to handle the transition from version 4 to version 6 are dual stack, tunneling, and header translation.

## 27.7 PRACTICE SET

### Exercises

1. An IPv6 packet consists of the base header and a TCP segment. The length of data is 320 bytes. Show the packet and enter a value for each field.
2. An IPv6 packet consists of a base header and a TCP segment. The length of data is 128,000 bytes (jumbo payload). Show the packet and enter a value for each field.

### Research Activity

3. Find out why there are two security protocols (AH and ESP) in IPv6.

CHAPTER

# 28

# *ICMPv6*

This is the last chapter in Part V of the book. After discussing the IPv6 addresses in Chapter 26 and the IPv6 datagrams in Chapter 27, we discuss the ICMPv6 protocol in this chapter. ICMPv6 is a combination of three protocols discussed for IPv4: ICMP, IGMP, and ARP. We first introduce ICMPv6 and divide its messages into four broad categories. We then briefly discuss the messages in each category.

## OBJECTIVES

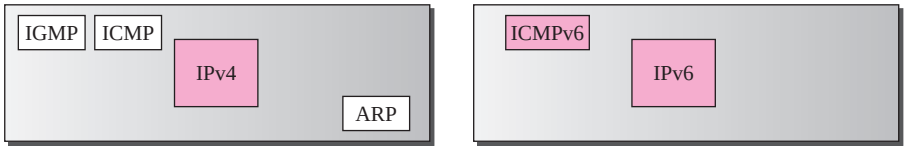
*The chapter has several objectives:*

- To introduce ICMPv6 and compare and contrast it with ICMPv4.
- To discuss error messages in ICMPv6 and compare and contrast them with the error messages in ICMPv4.
- To discuss informational messages in ICMPv6 and compare and contrast them with the informational messages in ICMPv4.
- To discuss neighbor discovery messages as part of the ND and IND protocol.
- To discuss group membership messages as part of the MLDv2 protocol.

## 28.1 INTRODUCTION

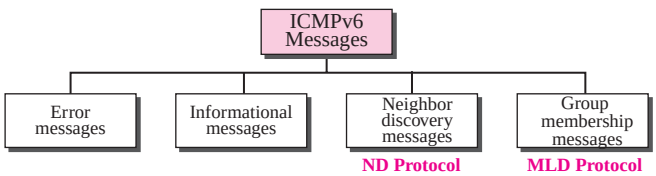
Another protocol that has been modified in version 6 of the TCP/IP protocol suite is ICMP. This new version, **Internet Control Message Protocol version 6 (ICMPv6),** follows the same strategy and purposes of version 4. ICMPv6, however, is more complicated than ICMPv4: some protocols that were independent in version 4 are now part of ICMPv6 and some new messages have been added to make it more useful. Figure 28.1 compares the network layer of version 4 to that of version 6. The ICMP, ARP, and IGMP protocols in version 4 are combined into one single protocol, ICMPv6.

**Figure 28.1** *Comparison of network layer in version 4 and version 6*




ICMPv6, like ICMPv4, is message-oriented; it uses messages to report errors, to get information, probe a neighbor, or manage multicast communication. However, a few other protocols are added to ICMPv6 to define the functionality and interpretation of the messages. Literature and RFCs use different strategies to group ICMPv6 messages. They define some of these messages as ICMPv6 messages and some as ND messages or MLD messages. We mention all of these messages as ICMPv6 messages, but we categorize them according to their functions and the roles they play. In each category, we also mention and describe the corresponding protocol that has been added for functionality and interpretation of the messages. Our reason for this taxonomy is that all messages have the same type of format and the message types are handled by ICMPv6 protocols. We believe that other protocols such as ND and MLD operate under the ICMPv6 protocol. With this justification in mind, we define the taxonomy of ICMPv6 messages as shown in Figure 28.2.

**Figure 28.2** *Taxonomy of ICMPv6 messages*

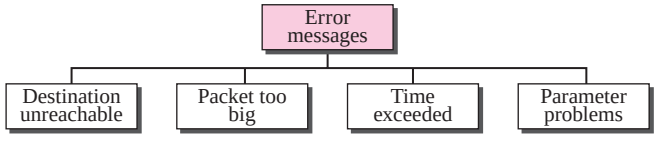


The figure shows that two groups of messages are sent and received under the control of ND protocol or MLD protocol.

## 28.2 ERROR MESSAGES

As we saw in our discussion of version 4, one of the main responsibilities of ICMP is to report errors. Four types of errors are handled: destination unreachable, packet too big, time exceeded, and parameter problems (see Figure 28.3). Note that the source-quenched message, which is used to control congestion in version 4, is eliminated in this version because the priority and flow label fields in IPv6 are supposed to take care of congestion. The redirection message has moved from the error-reporting category to the neighbor-discovery category, so we discuss it as part of the neighbor-discovery messages.

**Figure 28.3** *Error-reporting messages*

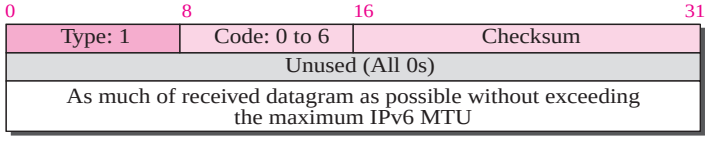



ICMPv6 forms an error packet, which is then encapsulated in an IPv6 datagram. This is delivered to the original source of the failed datagram.

### Destination-Unreachable Message

The concept of the destination unreachable message is the same as described for ICMPv4. When a router cannot forward a datagram or a host cannot deliver the content of the datagram to the upper layer protocol, the router or the host discards the datagram and sends a *destination-unreachable* error message to the source host. Figure 28.4 shows the format of the destination-unreachable message.

**Figure 28.4** *Destination-unreachable message*




The code field for this type specifies the reason for discarding the datagram and explains exactly what has failed:

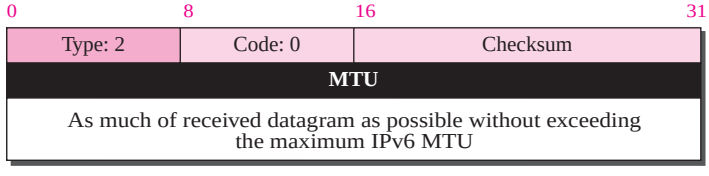
- **Code 0.** No path to destination.
- **Code 1.** Communication with the destination is administratively prohibited.

- **Code 2.** Beyond the scope of source address.
- **Code 3.** Destination address is unreachable.
- **Code 4.** Port unreachable.
- **Code 5.** Source address failed (filtering policy).
- **Code 6.** Reject route to destination.

## Packet-Too-Big Message

This is a new type of message added to version 6. Since IPv6 does not fragment at the router, if a router receives a datagram that is larger than the maximum transmission unit (MTU) size of the network through which the datagram should pass, two things happen. First, the router discards the datagram. Second, an ICMP error packet—a **packet-too-big message**—is sent to the source. Figure 28.5 shows the format of the packet. Note that there is only one code (0) and that the MTU field informs the sender of the maximum size packet accepted by the network.

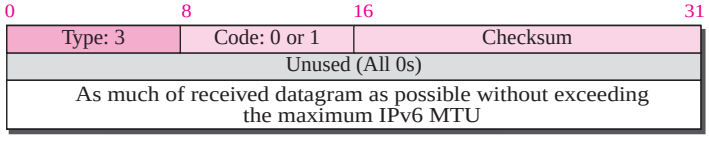
**Figure 28.5** *Packet-too-big message*




## Time-Exceeded Message

As we discussed in Chapter 9, a *time-exceeded* error message is generated in two cases: when the *time to live* value becomes zero and when not all fragments of a datagram have arrived in the time limit. The format of the *time-exceeded* message in version 6 is similar to the one in version 4. The only difference is that the type value has changed to 3. Figure 28.6 shows the format of the time-exceeded message.

**Figure 28.6** *Time-exceeded message*

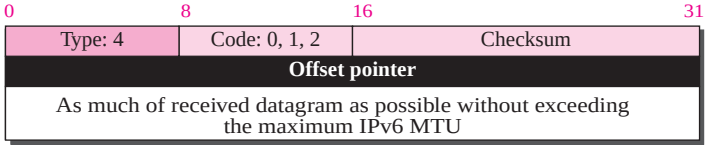



As in version 4, code 0 is used when the datagram is discarded by the router due to a hop-limit field value of zero. Code 1 is used when fragments of a datagram are discarded because other fragments have not arrived within the time limit.

### Parameter-Problem Message

As discussed in Chapter 9, any ambiguity in the header of the datagram can create serious problems as the datagram travels through the Internet. If a router or the destination host discovers any ambiguous or missing value in any field, it discards the datagram and sends a parameter-problem message to the source. The message in ICMPv6 is similar to its version 4 counterpart. However, the type value has been changed to 4 and the size of the offset pointer field has been increased to 4 bytes. There are also three different codes instead of two. Figure 28.7 shows the format of the parameter problem message.

**Figure 28.7** *Parameter-problem message*




The code field specifies the reason for discarding the datagram and the cause of failure:

- **Code 0.** Erroneous header field.
- **Code 1.** Unrecognized next header type.
- **Code 2.** Unrecognized IPv6 option.

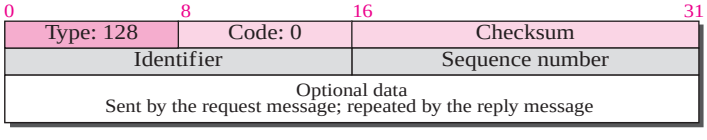
## 28.3 INFORMATIONAL MESSAGES

Two of the ICMPv6 messages can be categorized as informational messages: echo request and echo reply messages. As discussed in Chapter 9, the echo request and echo response messages are designed to check if two devices in the Internet can communicate with each other. A host or router can send an echo request message to another host; the receiving computer or router can reply using the echo response message.

### Echo-Request Message

The idea and format of the echo-request message is the same as the one in version 4. The only difference is the value for the type as shown in Figure 28.8.

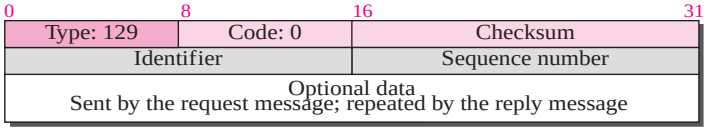
**Figure 28.8** *Echo-request message*



### Echo-Reply Message

The idea and format of the echo-reply message is the same as the one in version 4. The only difference is the value for the type as shown in Figure 28.9.

**Figure 28.9** *Echo-reply messages*




## 28.4 NEIGHBOR-DISCOVERY MESSAGES

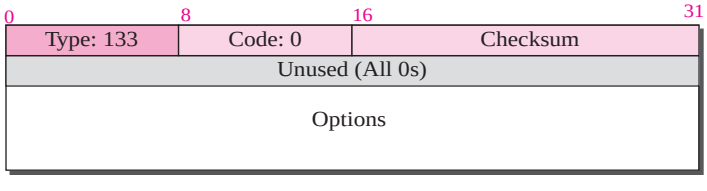
Several messages in the ICMPv6 have been redefined in ICMPv6 to handle the issue of neighbor discovery. Some new messages have also been added to provide extension. The most important issue is the definition of two new protocols that clearly define the functionality of these group messages: the **Neighbor-Discovery (ND) protocol** and the **Inverse-Neighbor-Discovery (IND) protocol.** These two protocols are used by nodes (hosts or routers) on the same link (network) for three main purposes:

1. Hosts use the ND protocol to find routers in the neighborhood that will forward packets for them.
2. Nodes use the ND protocol to find the link layer addresses of neighbors (nodes attached to the same network).
3. Nodes use the IND protocol to find the IPv6 addresses of the neighbor.

### Router-Solicitation Message

The idea behind the *router-solicitation* message is the same as in version 4. A host uses the router-solicitation message to find a router in the network that can forward an IPv6 datagram for the host. The only option that is so far defined for this message is the inclusion of physical (data link layer) address of the host to make the response easier for the router. The format of the message is shown in Figure 28.10. The type of this message is 133.

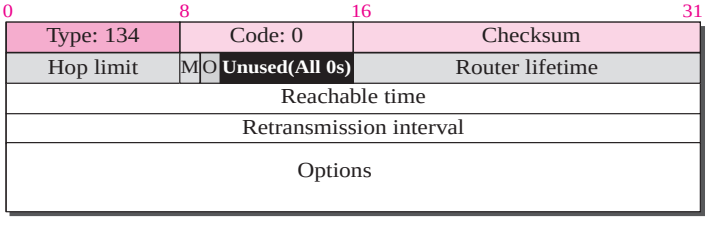
**Figure 28.10** *Router-solicitation message*



## Router-Advertisement Message

The *router-advertisement* message is sent by a router in response to a router solicitation message. Figure 28.11 shows the format of the router-advertisement message.

**Figure 28.11** *Router-advertisement message*



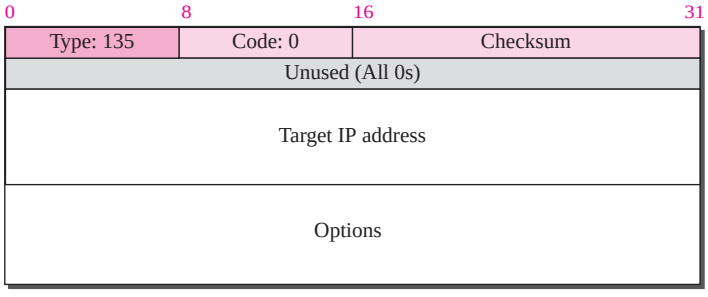

The fields are explained below:

- **Hop Limit.** This 8-bit field limits the number of hops that the requestor should use as the hop limit in its IPv6 datagram.
- **M.** This 1-bit field is the "manage address configuration" field. When this bit is set to 1, the host needs to use the administration configuration.
- **O.** This 1-bit field is the "other address configuration" field. When this bit is set to 1, the host needs to use the appropriate protocol for configuration.
- **Router Lifetime.** This 16-bit field defines the lifetime (in units of seconds) of the router as the default router. When the value of this field is 0, it means that the router is not a default router.
- **Reachable Time.** This 32-bit field defines the time (in units of seconds) that the router is reachable.
- **Retransmission Interval.** This 32-bit field defines the retransmission interval (in units of seconds).
- **Option.** Some possible options are the link layer address of the link from which the message is sent, the MTU of the link, and address prefix information.

## Neighbor-Solicitation Message

Figure 28.12 shows the format of **neighbor-solicitation message.** As previously mentioned, the network layer in version 4 contains an independent protocol called Address Resolution Protocol (ARP). In version 6, this protocol is eliminated, and its duties are included in ICMPv6. The neighbor solicitation message has the same duty as the ARP request message. This message is sent when a host or router has a message to send to a neighbor. The sender knows the IP address of the receiver, but needs the data link address of the receiver. The data link address is needed for the IP datagram to be encapsulated in a frame. The only option announces the sender data link address for the convenience of the receiver. The receiver can use the sender data link address to use a unicast response.

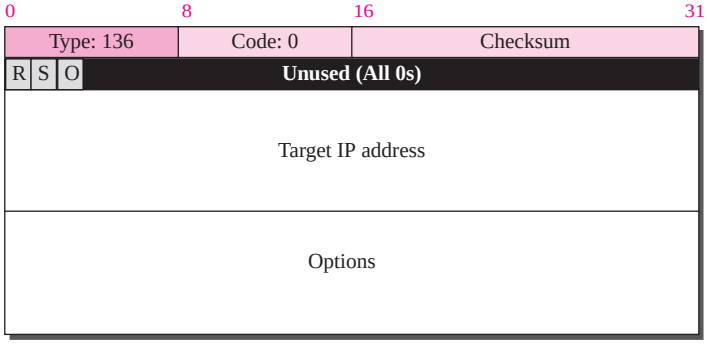
**Figure 28.12** *Neighbor-solicitation message*




## Neighbor-Advertisement Message

The **neighbor-advertisement message** is sent in response to the neighbor-solicitation message. This is equivalent to the ARP reply message in IPv4. Figure 28.13 shows the format of this message.

**Figure 28.13** *Neighbor-advertisement message*



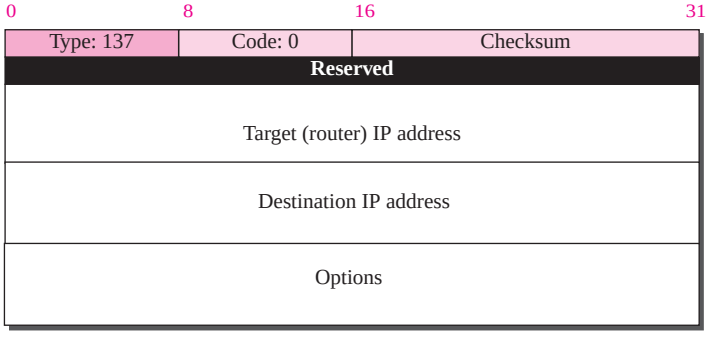

The fields are explained below:

- **R.** This 1-bit field is the "router" flag. When it is set to 1, it means the sender of this message is a router.
- **S.** This 1-bit field is the "solicitation" flag. When it is set to 1, it means that the sender is sending this advertisement in response to a neighbor solicitation. An advertisement can be sent by a host or router without solicitation.
- **O.** This 1-bit field is the "override" flag. When it is set, it means that the advertisement should override existing information in the cache.
- **Option.** The only possible option is the link layer address of the advertiser.

## Redirection Message

The purpose of the redirection message is the same as described for version 4. However, the format of the packet now accommodates the size of the IP address in version 6. Also, an option is added to let the host know the physical address of the target router (see Figure 28.14).

**Figure 28.14** *Redirection message*

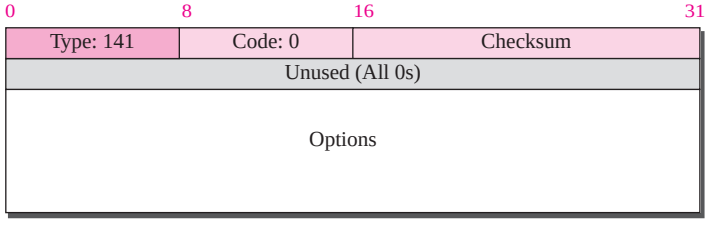



The possible option is the inclusion of the sender data link address and the part of the redirected IP header as long as the total size of the message does not exceed the MTU.

## Inverse-Neighbor-Solicitation Message

The **inverse-neighbor-solicitation message** is sent by a node that knows the link layer address of a neighbor, but not the neighbor's IP address. The message is encapsulated in an IPv6 datagram using an all-node multicast address. The sender must send the following two pieces of information in the option field: its link-layer address and the link-layer address of the target node. The sender can also include its IP address and the MTU value for the link. Figure 28.15 shows the format of this message.

**Figure 28.15** *Inverse-neighbor-solicitation message*

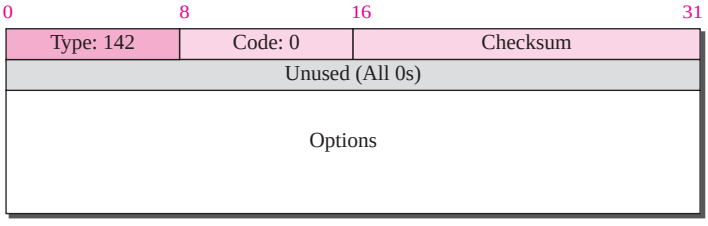



## Inverse-Neighbor-Advertisement Message

The **inverse-neighbor-advertisement message** is sent in response to the inverse-neighbor-discovery message. The sender of this message must include the link layer

address of the sender and the link layer address of the target node in the option section. Figure 28.16 shows the format of this message.

**Figure 28.16** *Inverse-neighbor-advertisement message*



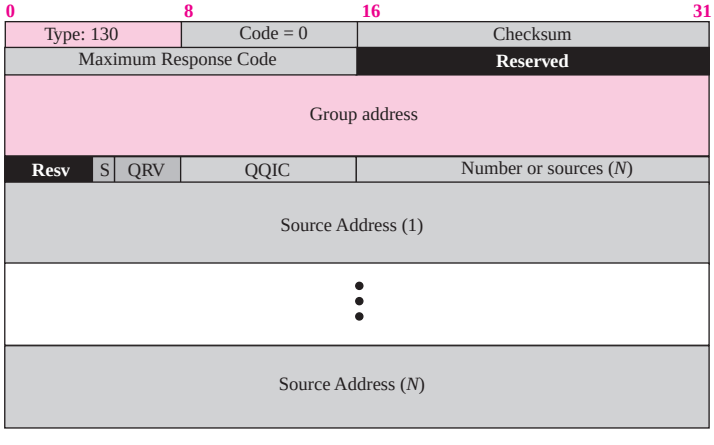

## 28.5 GROUP MEMBERSHIP MESSAGES

As we discuss in Chapter 12, the management of multicast delivery handling in IPv4 is given to the IGMPv3 protocol. In IPv6, this responsibility is given to the **Multicast Listener Delivery** protocol. MLDv1 is the counterpart to IGMPv2; MLDv2 is the counterpart to IGMPv3. The material discussed in this section is taken from RFC 3810. The idea is the same as we discussed in IGMPv3, but the sizes and formats of the messages have been changed to fit the larger multicast address size in IPv6. Like IGMPv3, MLDv2 has two types of messages: *membership-query message* and *membership-report message*. The first type can be divided into three subtypes: *general*, *group-specific*, and *group-and-source specific*.

### Membership-Query Message

A membership-query message is sent by a router to find active group members in the network. Figure 28.17 shows the format of this message.

**Figure 28.17** *Membership-query message format*

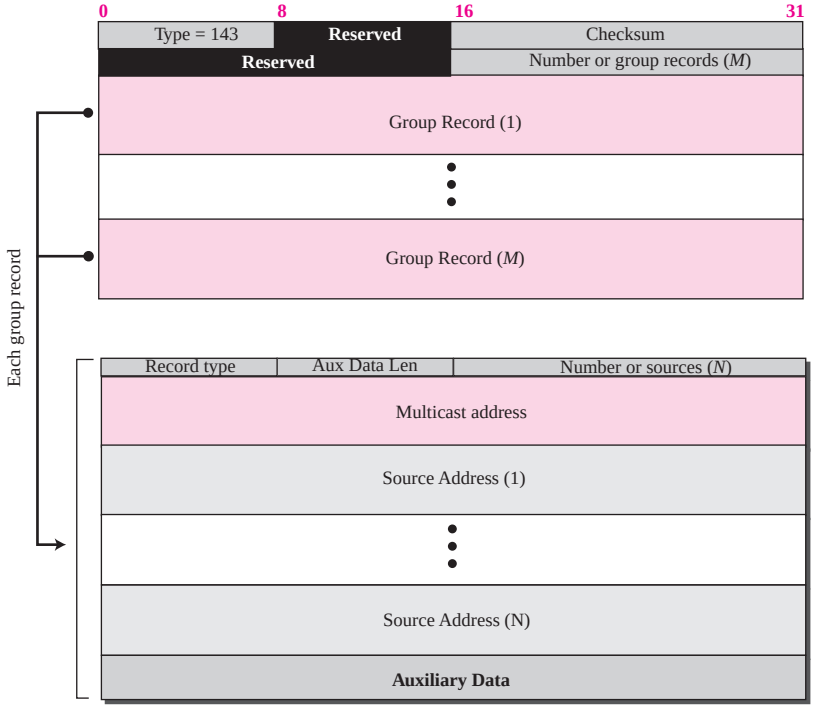


The fields are almost the same as the ones in IGMPv3 except that the size of the multicast address and the source address has been changed from 32 bits to 128 bits. Another noticeable change in the field size is in the *maximum response code* field, in which the size has been changed from 8 bits to 16 bits. We will discuss this field shortly. Also note that the format of the first 8 bytes matches the format for other ICMPv6 packets because MLDv2 is considered to be part of ICMPv6.

## Membership-Report Message

Figure 28.18 shows the format of a membership report message format.

**Figure 28.18** *Membership-report message format*




Note that the format of the membership report in MLDv2 is exactly the same as the one in IGMPv3 except that the sizes of the fields are changed because of the address size. In particular, the record type is the same as the one defined for IGMPv3 (types 1 to 6).

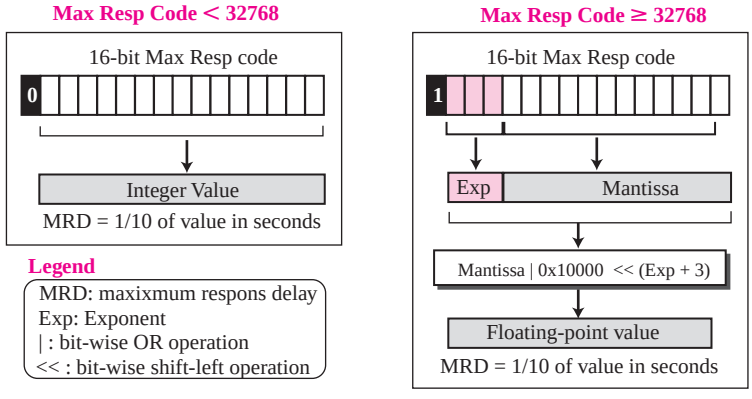
## Functionality

MDLv2 protocol behaves in the same way as IGMPv3. However, there are a few differences that we discuss here.

### *Calculation of Maximum Response Time*

As we mentioned, the size of the Max Resp Code in MLV2 is twice the size of the same field in IGMPv3. For this reason, the calculation of maximum response time is slightly different in this protocol as shown in Figure 28.19.

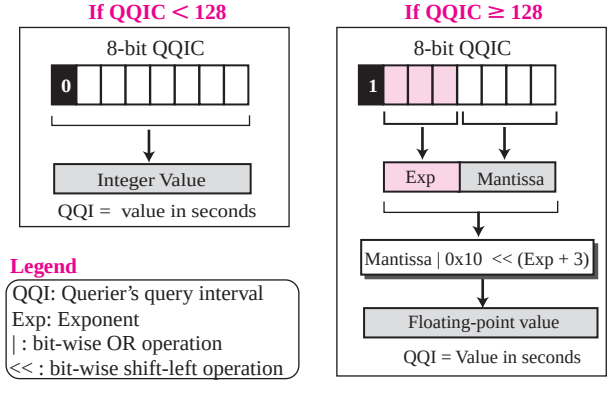
**Figure 28.19** *Calculation of maximum response time*




### *Calculation of Query Interval*

The calculation of query interval follows the same process as the calculation of maximum response delay; it is calculated from the value of the QQIC field as shown in Figure 28.20.

**Figure 28.20** *Calculation of maximum response time*



## 28.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give some coverage of ICMPv6. We recommend [Com 06], [Los 04], and [Koz 05].

### RFCs

Several RFCs show various updates of ICMPv6, including RFC 2461, RFC 2894, RFC 3122, RFC 3810, RFC 4443, and RFC 4620.

## 28.7 KEY TERMS

Internetworking Control Message Protocol, version 6 (ICMPv6)
Inverse-Neighbor-Discovery (IND) protocol
inverse-neighbor-advertisement message
inverse-neighbor-solicitation message
Multicast Listening Delivery protocol
Neighbor-Discovery (ND) protocol
neighbor-advertisement message
neighbor-solicitation message
packet-too-big message

## 28.8 SUMMARY

- ICMPv6, like ICMPv4, is message-oriented; it uses messages to report errors, to get information, probe a neighbor, or manage multicast communication. However, a few other protocols are added to ICMPv6 to define the functionality and interpretation of the messages.
- We have divided all messages in the ICMPv6 into four categories: error messages, informational messages, neighbor-discovery messages, and group-membership messages.
- Four types of error messages have been discussed: destination-unreachable, packet-too-big, time-exceeded, and parameter-problems.
- Two types of informational messages have been discussed: echo-request and echo-response.
- We discussed seven neighbor-discovery messages. The first five, router-solicitation, router-advertisment, neighbor-solicitation, neighbor-advertisment, and redirect are under the control of the ND protocol. The last two messages, inverse-neighbor-solicitation and inverse-neighbor-advertisment are under the control of the IND protocol.
- We discussed two group management messages: membership-query and membership-report. They are under the control of MDLv2 protocol.

## 28.9 PRACTICE SET

### Exercises

1. Which ICMP messages contain part of the IP datagram? Why is this needed?
2. Make a table to compare and contrast error-reporting messages in ICMPv6 with error-reporting messages ICMPv4.
3. Make a table to compare and contrast informational messages in ICMPv6 with informational messages in ICMPv4.
4. Make a table to compare and contrast neighbor-discovery messages in ICMPv6 with the corresponding messages in version 4.
5. Make a table to compare and contrast inverse neighbor-discovery messages in ICMPv6 with the corresponding messages in version 4.
6. Make a table to compare and contrast group-membership messages in ICMPv6 with the corresponding messages in version 4.
7. Calculate the value of maximum response delay (in seconds) for each of the following maximum response code values (see Figure 28.19).
   a. 22000
   b. 43000
8. Calculate the value of QQI for each of the following QQIC values (see Figure 28.20).
   a. 78
   b. 202

### Research Activities

9. Use RFCs 3810 and 4604 to learn more about MLDv2.
10. Use RFC 2461 to learn more about the role of ND protocol.
11. Use RFC 3122 to learn more about the role of IND protocol.
12. Use RFC 2894 to learn more about *router renumbering* for IPv6.

# PART 6

# Security

CHAPTER

# 29

# *Cryptography and Network Security*

The topic of *cryptography* and *network security* is very broad and involves some specific areas of mathematics such as number theory. In this chapter, we try to give a very simple introduction to this topic to prepare the background for the next chapter, in which we discuss Internet security. Our goal is to briefly discuss the general issues related to cryptography and network security without being involved in the mathematical details behind each issue.

## OBJECTIVES

*The chapter has several objectives:*

- To introduce security goals and to discuss the types of attacks that threaten these goals.
- To introduce traditional ciphers as symmetric-key ciphers to create the background for understanding modern symmetric-key ciphers.
- To introduce the elements of modern block ciphers and show an example of a modern block cipher in which these elements are used.
- To discuss the general idea behind asymmetric-key ciphers and introduce one common cipher in this category.
- To discuss message integrity and show how to use a cryptographic hash function to create a message digest.
- To introduce the idea of message authentication and show how a message digest combined with a secret can authenticate the sender.
- To show how the idea of digital signatures can be used to authenticate a message using a pair of private-public keys.
- To introduce the idea of entity authentication and show some simple schemes using either a secret key or a pair of private-public keys.
- To show how secret keys in symmetric-key cryptography and how public keys in asymmetric-key cryptography can be distributed and managed using KDCs or certificate authorities (CAs).

## 29.1 INTRODUCTION

We are living in the information age. We need to keep information about every aspect of our lives. In other words, information is an asset that has a value like any other asset. As an asset, information needs to be secured from attacks. To be secured, information needs to be hidden from unauthorized access (*confidentiality*), protected from unauthorized change (*integrity*), and available to an authorized entity when it is needed (*availability*).

During the last three decades, computer networks created a revolution in the use of information. Information is now distributed. Authorized people can send and retrieve information from a distance using computer networks. Although the three above-mentioned requirements—confidentiality, integrity, and availability—have not changed, they now have some new dimensions. Not only should information be confidential when it is stored; there should also be a way to maintain its confidentiality when it is transmitted from one computer to another.

In this section, we first discuss the three major goals of information security. We then see how attacks can threaten these three goals. We then discuss the security services in relation to these security goals. Finally we define two techniques to implement the security goals and prevent attacks.

### Security Goals

Let us first discuss three security goals: confidentiality, integrity, and availability.

#### *Confidentiality*

**Confidentiality** is probably the most common aspect of information security. We need to protect our confidential information. An organization needs to guard against those malicious actions that endanger the confidentiality of its information. Confidentiality not only applies to the storage of the information, it also applies to the transmission of information. When we send a piece of information to be stored in a remote computer or when we retrieve a piece of information from a remote computer, we need to conceal it during transmission.

#### *Integrity*

Information needs to be changed constantly. In a bank, when a customer deposits or withdraws money, the balance of her account needs to be changed. **Integrity** means that changes need to be done only by authorized entities and through authorized mechanisms. Integrity violation is not necessarily the result of a malicious act; an interruption in the system, such as a power surge, may also create unwanted changes in some information.

#### *Availability*

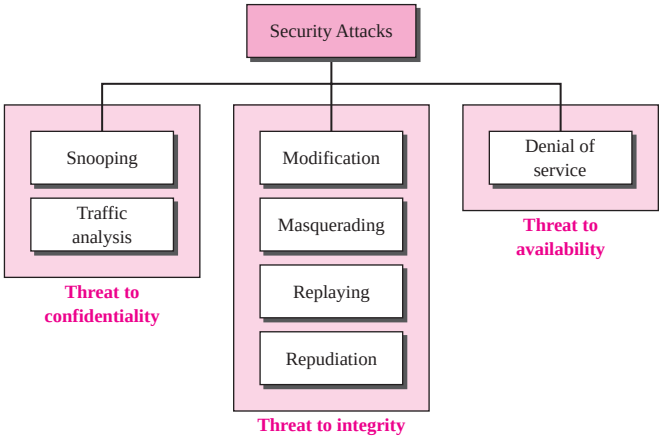
The third component of information security is **availability.** The information created and stored by an organization needs to be available to authorized entities. Information is

useless if it is not available. Information needs to be constantly changed, which means it must be accessible to authorized entities. The unavailability of information is just as harmful for an organization as the lack of confidentiality or integrity. Imagine what would happen to a bank if the customers could not access their accounts for transactions.

## Attacks

Our three goals of security—confidentiality, integrity, and availability—can be threatened by security **attacks.** Although the literature uses different approaches to categorizing the attacks, we divide them into three groups related to the security goals. Figure 29.1 shows the taxonomy.

**Figure 29.1** *Taxonomy of attacks with relation to security goals*




### *Attacks Threatening Confidentiality*

In general, two types of attacks threaten the confidentiality of information: **snooping** and **traffic analysis.**

**Snooping** Snooping refers to unauthorized access to or interception of data. For example, a file transferred through the Internet may contain confidential information. An unauthorized entity may intercept the transmission and use the contents for her own benefit. To prevent snooping, the data can be made nonintelligible to the intercepter by using encryption techniques discussed in this book.

**Traffic Analysis** Although encipherment of data may make it nonintelligible for the intercepter, she can obtain some other type information by monitoring online traffic. For example, she can find the electronic address (such as the e-mail address) of the sender or the receiver. She can collect pairs of requests and responses to help her guess the nature of the transaction.

### *Attacks Threatening Integrity*

The integrity of data can be threatened by several kinds of attacks: **modification, masquerading, replaying,** and **repudiation.**

**Modification** After intercepting or accessing information, the attacker modifies the information to make it beneficial to herself. For example, a customer sends a message to a bank to do some transaction. The attacker intercepts the message and changes the type of transaction to benefit herself. Note that sometimes the attacker simply deletes or delays the message to harm the system or to benefit from it.

**Masquerading** Masquerading, or spoofing, happens when the attacker impersonates somebody else. For example, an attacker might steal the bank card and PIN of a bank customer and pretend that she is that customer. Sometimes the attacker pretends to be the receiver entity. For example, a user tries to contact a bank, but another site pretends that it is the bank and obtains some information from the user.

**Replaying** Replaying is another attack. The attacker obtains a copy of a message sent by a user and later tries to replay it. For example, a person sends a request to her bank to ask for payment to the attacker, who has done a job for her. The attacker intercepts the message and sends it again to receive another payment from the bank.

**Repudiation** This type of attack is different from others because it is performed by one of the two parties in the communication: the sender or the receiver. The sender of the message might later deny that she has sent the message; the receiver of the message might later deny that he has received the message. An example of denial by the sender would be a bank customer asking her bank to send some money to a third party but later denying that she has made such a request. An example of denial by the receiver could occur when a person buys a product from a manufacturer and pays for it electronically, but the manufacturer later denies having received the payment and asks to be paid.

### *Attacks Threatening Availability*

We mention only one attack threatening availability: **denial of service.**

**Denial of Service** Denial of service (DoS) is a very common attack. It may slow down or totally interrupt the service of a system. The attacker can use several strategies to achieve this. She might send so many bogus requests to a server that the server crashes because of the heavy load. The attacker might intercept and delete a server's response to a client, making the client believe that the server is not responding. The attacker may also intercept requests from the clients, causing the clients to send requests many times and overload the system.

## Services

ITU-T defines some security services to achieve security goals and prevent attacks. Each of these services is designed to protect one or more attacks while maintaining security goals.

## Techniques

The actual implementation of security goals needs some techniques. Two techniques are prevalent today: one is very general (cryptography) and one is specific (steganography).

### *Cryptography*

Some security services can be implemented using cryptography. **Cryptography,** a word with Greek origins, means "secret writing." However, we use the term to refer to the

science and art of transforming messages to make them secure and immune to attacks. Although in the past *cryptography* referred only to the **encryption** and **decryption** of messages using secret keys, today it is defined as involving three distinct mechanisms: symmetric-key encipherment, asymmetric-key encipherment, and hashing. We will discuss all these techniques later in the chapter.

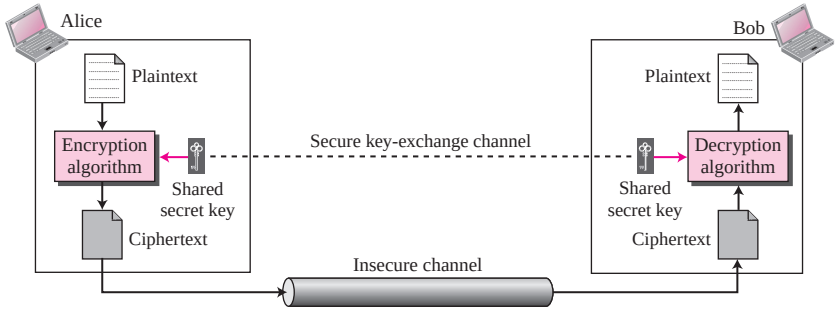
### *Steganography*

Although this chapter and the next are based on cryptography as a technique for implementing security services, another technique that was used for secret communication in the past is being revived at the present time: steganography. The word **steganography,** with origin in Greek, means "covered writing," in contrast with cryptography, which means "secret writing." Cryptography means concealing the contents of a message by enciphering; steganography means concealing the message itself by covering it with something else. We leave the discussion of steganography to some books dedicated to this topic.

## 29.2 TRADITIONAL CIPHERS

We now look at the first goal of security, confidentiality. Confidentiality can be achieved using ciphers. Traditional ciphers are called **symmetric-key ciphers** (or secret-key ciphers) because the same key is used for encryption and decryption and the key can be used for bidirectional communication. Figure 29.2 shows the general idea behind a symmetric-key cipher.

**Figure 29.2** *General idea of a traditional cipher*




**Symmetric-key ciphers are also called secret-key ciphers.**

In Figure 29.2, an entity, Alice, can send a message to another entity, Bob, over an insecure channel with the assumption that an adversary, Eve, cannot understand the contents of the message by simply eavesdropping over the channel.

The original message from Alice to Bob is called **plaintext;** the message that is sent through the channel is called the **ciphertext.** To create the ciphertext from the plaintext, Alice uses an **encryption algorithm** and a **shared secret key.** To create the plaintext from ciphertext, Bob uses a **decryption algorithm** and the same secret key.

We refer to encryption and decryption algorithms as **ciphers.** A **key** is a set of values (numbers) that the cipher, as an algorithm, operates on.

Note that the symmetric-key encipherment uses a single key (the key itself may be a set of values) for both encryption and decryption. In addition, the encryption and decryption algorithms are inverses of each other. If P is the plaintext, C is the ciphertext, and K is the key, the encryption algorithm $E_k(x)$ creates the ciphertext from the plaintext; the decryption algorithm $D_k(x)$ creates the plaintext from the ciphertext. We assume that $E_k(x)$ and $D_k(x)$ are inverses of each other: they cancel the effect of each other if they are applied one after the other on the same input. We have

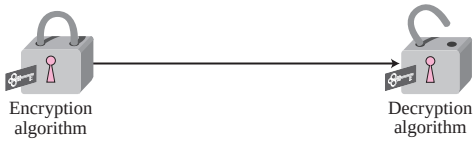
**Encryption: $C = E_k(P)$** **Decryption: $P = D_k(C)$**

in which, $D_k(E_k(x)) = E_k(D_k(x)) = x$. We need to emphasize that it is better to make the encryption and decryption public but keep the shared key secret. This means that Alice and Bob need another channel, a secured one, to exchange the secret key. Alice and Bob can meet once and exchange the key personally. The secured channel here is the face-to-face exchange of the key. They can also trust a third party to give them the same key. They can create a temporary secret key using another kind of cipher—asymmetric-key ciphers—which will be described later.

## Key

Encryption can be thought of as locking the message in a box; decryption can be thought of as unlocking the box. In symmetric-key encipherment, the same key locks and unlocks as shown in Figure 29.3. Later chapters show that the *asymmetric-key* encipherment needs two keys, one for locking and one for unlocking.

**Figure 29.3** *Symmetric-key encipherment as locking and unlocking with the same key*




## Substitution Ciphers

We can divide traditional symmetric-key ciphers into two broad categories: substitution ciphers and transposition ciphers. A **substitution cipher** replaces one symbol with another. If the symbols in the plaintext are alphabetic characters, we replace one character with another. For example, we can replace letter A with letter D, and letter T with letter Z. If the symbols are digits (0 to 9), we can replace 3 with 7, and 2 with 6.

**A substitution cipher replaces one symbol with another.**

Substitution ciphers can be categorized as either monoalphabetic ciphers or polyalphabetic ciphers.

### *Monoalphabetic Ciphers*

In a **monoalphabetic cipher,** a character (or a symbol) in the plaintext is always changed to the same character (or symbol) in the ciphertext regardless of its position in the text. For example, if the algorithm says that letter A in the plaintext is changed to letter D, every letter A is changed to letter D. In other words, the relationship between letters in the plaintext and the ciphertext is one-to-one.

The simplest monoalphabetic cipher is the **additive cipher** (or **shift cipher**). Assume that the plaintext consists of lowercase letters (a to z), and that the ciphertext consists of uppercase letters (A to Z). To be able to apply mathematical operations on the plaintext and ciphertext, we assign numerical values to each letter (lower- or uppercase), as shown in Figure 29.4.

**Figure 29.4** *Representation of plaintext and ciphertext characters in modulo 26*

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Plaintext → | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
| Ciphertext → | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| Value → | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

In Figure 29.4 each character (lowercase or uppercase) is assigned an integer in modulo 26. The secret key between Alice and Bob is also an integer in modulo 26. The encryption algorithm adds the key to the plaintext character; the decryption algorithm subtracts the key from the ciphertext character. All operations are done in modulo 26.

**In additive cipher, the plaintext, ciphertext, and key are integers in modulo 26.**

Historically, additive ciphers are called shift ciphers because encryption algorithm can be interpreted as "shift *key* characters down" and the encryption algorithm can be interpreted as "shift *key* characters up". Julius Caesar used an additive cipher, with a key of 3 to communicate with his officers. For this reason, additive ciphers are sometimes referred to as the **Caesar cipher.**

### Example 29.1

Use the additive cipher with key = 15 to encrypt the message "hello".

**Solution**

We apply the encryption algorithm to the plaintext, character by character:

| | | |
|---|---|---|
| Plaintext: h → 07 | Encryption: (07 + 15) mod 26 | Ciphertext: 22 → W |
| Plaintext: e → 04 | Encryption: (04 + 15) mod 26 | Ciphertext: 19 → T |
| Plaintext: l → 11 | Encryption: (11 + 15) mod 26 | Ciphertext: 00 → A |
| Plaintext: l → 11 | Encryption: (11 + 15) mod 26 | Ciphertext: 00 → A |
| Plaintext: o → 14 | Encryption: (14 + 15) mod 26 | Ciphertext: 03 → D |

The result is "WTAAD". Note that the cipher is monoalphabetic because two instances of the same plaintext character (*ls*) are encrypted as the same character (*A*).

### Example 29.2

Use the additive cipher with key = 15 to decrypt the message "WTAAD".

**Solution**

We apply the decryption algorithm to the plaintext character by character:

| | | |
|---|---|---|
| Ciphertext: W → 22 | Decryption: (22 – 15) mod 26 | Plaintext: 07 → h |
| Ciphertext: T → 19 | Decryption: (19 – 15) mod 26 | Plaintext: 04 → e |
| Ciphertext: A → 00 | Decryption: (00 – 15) mod 26 | Plaintext: 11 → l |
| Ciphertext: A → 00 | Decryption: (00 – 15) mod 26 | Plaintext: 11 → l |
| Ciphertext: D → 03 | Decryption: (03 – 15) mod 26 | Plaintext: 14 → o |

The result is "hello". Note that the operation is in modulo 26, which means that we need to add 26 to a negative result (for example -15 becomes 11).

Additive ciphers are vulnerable to attacks using exhaustive key searches (brute-force attacks). The key domain of the additive cipher is very small; there are only 26 keys. However, one of the keys, zero, is useless (the ciphertext is the same as the plaintext). This leaves only 25 possible keys. Eve can easily launch a brute-force attack on the ciphertext.

Because additive ciphers have small key domains, they are very vulnerable to attack. A better solution is to create a mapping between each plaintext character and the corresponding ciphertext character. Alice and Bob can agree on a table showing the mapping for each character. Figure 29.5 shows an example of such a mapping.

**Figure 29.5** *An example key for monoalphabetic substitution cipher*

| Plaintext → | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ciphertext → | N | O | A | T | R | B | E | C | F | U | X | D | Q | G | Y | L | K | H | V | I | J | M | P | Z | S | W |

### Example 29.3

We can use the key in Figure 29.5 to encrypt the message

this message is easy to encrypt but hard to find the key

The ciphertext is

ICFVQRVVNEFVRNVSIYRGAHSLIOJICNHTIYBFGTICRXRS

### *Polyalphabetic Ciphers*

In **polyalphabetic substitution,** each occurrence of a character may have a different substitute. The relationship between a character in the plaintext to a character in the

ciphertext is one-to-many. For example, "a" could be enciphered as "D" in the beginning of the text, but as "N" at the middle. Polyalphabetic ciphers have the advantage of hiding the letter frequency of the underlying language. Eve cannot use single-letter frequency statistic to break the ciphertext.

To create a **polyalphabetic cipher,** we need to make each ciphertext character dependent on both the corresponding plaintext character and the position of the plaintext character in the message. This implies that our key should be a stream of subkeys, in which each subkey depends somehow on the position of the plaintext character that uses that subkey for encipherment. In other words, we need to have a key stream $k = (k_1, k_2, k_3, \ldots)$ in which $k_i$ is used to encipher the $i$th character in the plaintext to create the $i$th character in the ciphertext.

To see the position dependency of the key, let us discuss a simple polyalphabetic cipher called the **autokey cipher.** In this cipher, the key is a stream of subkeys, in which each subkey is used to encrypt the corresponding character in the plaintext. The first subkey is a predetermined value secretly agreed upon by Alice and Bob. The second subkey is the value of the first plaintext character (between 0 and 25). The third subkey is the value of the second plaintext. And so on.

$$P = P_1P_2P_3 \ldots \qquad C = C_1C_2C_3\ldots \qquad k = (k_1, P_1, P_2, \ldots)$$
$$\text{Encryption: } C_i = (P_i + k_i) \bmod 26 \qquad \text{Decryption: } P_i = (C_i - k_i) \bmod 26$$

The name of the cipher, *autokey*, implies that the subkeys are automatically created from the plaintext cipher characters during the encryption process.

## Transposition Ciphers

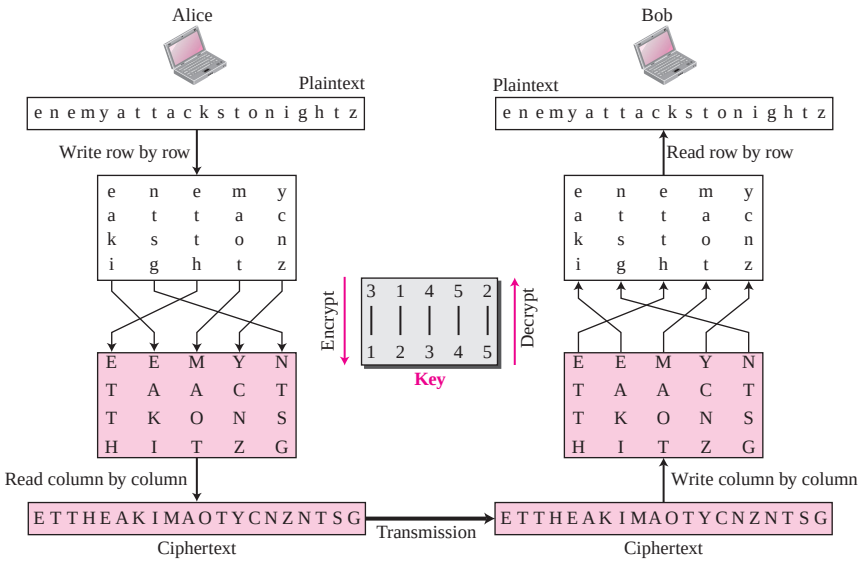
A **transposition cipher** does not substitute one symbol for another, instead it changes the location of the symbols. A symbol in the first position of the plaintext may appear in the tenth position of the ciphertext. A symbol in the eighth position in the plaintext may appear in the first position of the ciphertext. In other words, a transposition cipher reorders (transposes) the symbols.

**A transposition cipher reorders symbols.**

Suppose Alice wants to secretly send the message "Enemy attacks tonight" to Bob. The encryption and decryption is shown in Figure 29.6.

The first table is created by Alice writing the plaintext row by row. The columns are permuted using a key. The ciphertext is created by reading the second table column by column. Bob does the same three steps in the reverse order. He writes the ciphertext column by column into the first table, permutes the columns, and then reads the second table row by row. Note that the same key is used for encryption and decryption, but the algorithm uses the key in reverse order.

**Figure 29.6** *Transposition cipher*




## Stream and Block Ciphers

The literature divides the symmetric ciphers into two broad categories: stream ciphers and block ciphers.

### *Stream Cipher*

In a **stream cipher,** encryption and decryption are done one symbol (such as a character or a bit) at a time. We have a plaintext stream, a ciphertext stream, and a key stream. Call the plaintext stream P, the ciphertext stream C, and the key stream K.

$$P = P_1P_2P_3, \ \ldots \qquad C = C_1C_2C_3, \ \ldots \qquad K = (k_1, k_2, k_3, \ \ldots)$$
$$C_1 = E_{k1}(P_1) \qquad C_2 = E_{k2}(P_2) \qquad C_3 = E_{k3}(P_3) \ \ldots$$

### *Block Ciphers*

In a **block cipher,** a group of plaintext symbols of size $m$ ($m > 1$) are encrypted together, creating a group of ciphertext of the same size. Based on the definition, in a block cipher, a single key is used to encrypt the whole block even if the key is made of multiple values. In a block cipher, a ciphertext block depends on the whole plaintext block.

### *Combination*

In practice, blocks of plaintext are encrypted individually, but they use a stream of keys to encrypt the whole message block by block. In other words, the cipher is a block

cipher when looking at the individual blocks, but it is a stream cipher when looking at the whole message considering each block as a single unit. Each block uses a different key that may be generated before or during the encryption process. Examples of this will appear in later chapters.

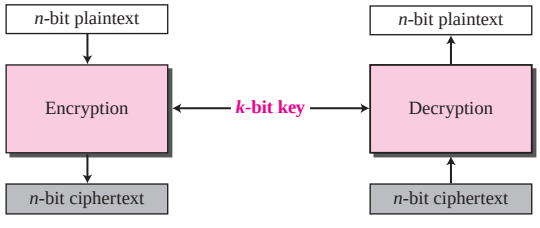
## 29.3 MODERN CIPHERS

The traditional symmetric-key ciphers that we have studied so far are **character-oriented ciphers.** With the advent of the computer, we need **bit-oriented ciphers.** This is because the information to be encrypted is not just text; it can also consist of numbers, graphics, audio, and video data. It is convenient to convert these types of data into a stream of bits, to encrypt the stream, and then to send the encrypted stream. In addition, when text is treated at the bit level, each character is replaced by 8 (or 16) bits, which means that the number of symbols becomes 8 (or 16) times larger. Mixing a larger number of symbols increases security. A modern block cipher can be either a block cipher or a stream cipher.

### Modern Block Ciphers

A symmetric-key **modern block cipher** encrypts an *n*-bit block of plaintext or decrypts an *n*-bit block of ciphertext. The encryption or decryption algorithm uses a *k*-bit key. The decryption algorithm must be the inverse of the encryption algorithm, and both operations must use the same secret key so that Bob can retrieve the message sent by Alice. Figure 29.7 shows the general idea of encryption and decryption in a modern block cipher.

**Figure 29.7** *A modern block cipher*



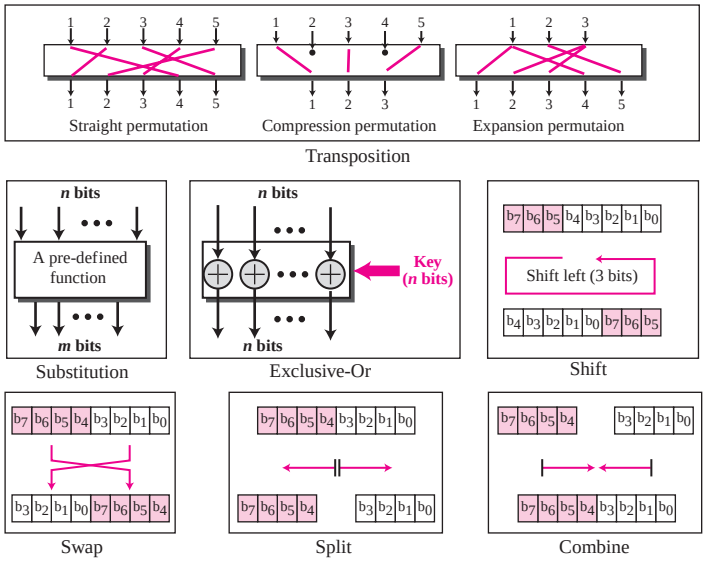

If the message has fewer than *n* bits, padding must be added to make it an *n*-bit block; if the message has more than *n* bits, it should be divided into *n*-bit blocks and the appropriate padding must be added to the last block if necessary. The common values for *n* are 64, 128, 256, and 512 bits.

#### *Components of a Modern Block Cipher*

Modern block ciphers are substitution ciphers when seen as a whole block. However, modern block ciphers are not designed as a single unit. To provide an attack-resistant

cipher, a modern block cipher is made of a combination of transposition units (sometimes called P-boxes), substitution units (sometimes called S-boxes), and exclusive-or operations, shifting elements, swapping elements, splitting elements, and combining elements. Figure 29.8 shows the components of a modern block cipher.

**Figure 29.8** *Components of a modern block cipher*




A **P-box** (permutation box) parallels the traditional transposition cipher for characters, but it transposes bits. We can find three types of P-boxes in modern block ciphers: straight P-boxes, expansion P-boxes, and compression P-boxes. An **S-box** (substitution box) can be thought of as a miniature substitution cipher, but it substitutes bits. Unlike the traditional substitution cipher, an S-box can have a different number of inputs and outputs. An important component in most block ciphers is the *exclusive-or* operation, in which the output is 0 if the two inputs are the same, and the output is 1 if the two inputs are different. In modern block ciphers, we use *n* exclusive-or operations to combine an *n*-bit data piece with an *n*-bit key. An exclusive-or operation is normally the only unit where the key is applied.

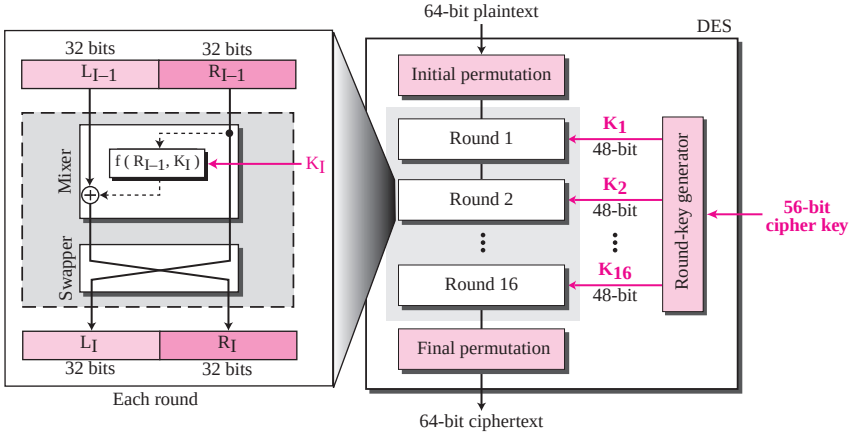
Another component found in some modern block ciphers is the **circular shift operation.** Shifting can be to the left or to the right. The circular left-shift operation shifts each bit in an *n*-bit word *k* positions to the left; the leftmost *k* bits are removed from the left and become the rightmost bits. The **swap operation** is a special case of the circular shift operation where the number of shifted bits $k = n/2$.

Two other operations found in some block ciphers are split and combine. The **split operation** splits an *n*-bit word in the middle, creating two equal-length words. The **combine operation** normally concatenates two equal-length words to create an *n*-bit word.

## Data Encryption Standard (DES)

As an example of a modern block cipher, let us discuss the **Data Encryption Standard (DES)**. Figure 29.9 shows the elements of DES cipher at the encryption site.

**Figure 29.9** *General structure of DES*




At the encryption site, DES takes a 64-bit plaintext and creates a 64-bit ciphertext; at the decryption site, DES takes a 64-bit ciphertext and creates a 64-bit block of plaintext. The same 56-bit cipher key is used for both encryption and decryption.

The initial permutations takes a 64-bit input and permutes them according to a predefined rule. The final permutation is the inverse of the initial permutation. These two permutations cancel the effect of each other. In other words, if the rounds are eliminated from the structures, the ciphertext is the same as the plaintext.

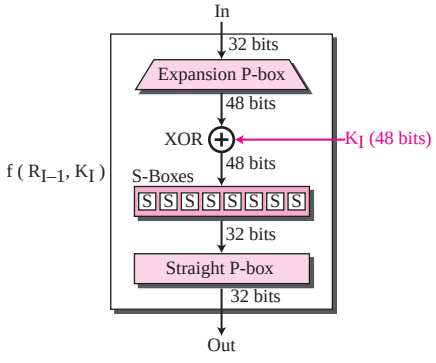
### *Rounds*

DES uses 16 rounds. Each round of DES is an invertible transformation, as shown in Figure 29.9. The round takes $L_{I-1}$ and $R_{I-1}$ from the previous round (or the initial permutation box) and creates $L_I$ and $R_I$, which go to the next round (or final permutation box). Each round can have up to two cipher elements (mixer and swapper). Each of these elements is invertible. The swapper is obviously invertible. It swaps the left half of the text with the right half. The mixer is invertible because of the XOR operation. All noninvertible elements are collected inside the function $\boldsymbol{f}\,(R_{I-1}, K_I)$.

### *DES Function*

The heart of DES is the DES function. The DES function applies a 48-bit key to the rightmost 32 bits ($R_{I-1}$) to produce a 32-bit output. This function is made up of four sections: an expansion P-box, an exclusive-OR component (that adds key), a group of S-boxes, and a straight P-box, as shown in Figure 29.10.

**Figure 29.10** *DES function*



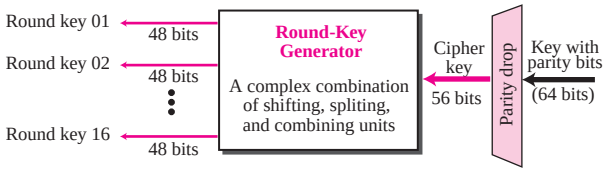

Since $R_{I-1}$ is a 32-bit input and $K_I$ is a 48-bit key, we first need to expand $R_{I-1}$ to 48 bits. This expansion permutation follows a predetermined rule.

After the expansion permutation, DES uses the XOR operation on the expanded right section and the round key. The S-boxes do the real mixing. DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output.The last operation in the DES function is a straight permutation with a 32-bit input and a 32-bit output.

### *Key Generation*

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. However, the cipher key is normally given as a 64-bit key in which 8 extra bits are the parity bits, which are dropped before the actual key-generation process, as shown in Figure 29.11.

**Figure 29.11** *Key generation*




### Example 29.4

We choose a random plaintext block, a random key, and a computer program to determine what the ciphertext block would be (all in hexadecimal):

| Plaintext: | **Key:** | CipherText: |
|---|---|---|
| **123456ABCD132536** | **AABB09182736CCDD** | **C0B7A8D05F3A829C** |

### Example 29.5

To check the effectiveness of DES, when a single bit is changed in the input, let us use two different plaintexts with only one single bit difference. The two ciphertexts are completely different without even changing the key:

| Plaintext: | Key: | Ciphertext: |
|---|---|---|
| 0000000000000000 | 22234512987ABB23 | 4789FD476E82A5F1 |
| Plaintext: | Key: | Ciphertext: |
| 0000000000000001 | 22234512987ABB23 | 0A4ED5C15A63FEA3 |

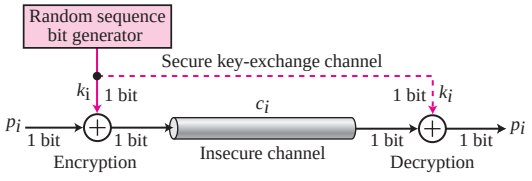
Although the two plaintext blocks differ only in the rightmost bit, the ciphertext blocks differ in 29 bits.

## Modern Stream Ciphers

In addition to modern block ciphers, we can also use modern stream ciphers. Similar differences exist between modern stream ciphers and modern block ciphers. In a **modern stream cipher,** encryption and decryption are done $r$ bits at a time. We have a plaintext bit stream $P = p_n \cdots p_2 p_1$, a ciphertext bit stream $C = c_n \cdots c_2 c_1$, and a key bit stream $K = k_n \cdots k_2 k_1$, in which $p_i$, $c_i$, and $k_i$ are $r$-bit words. Encryption is $c_i = E(k_i, p_i)$, and decryption is $p_i = D(k_i, c_i)$. Stream ciphers are faster than block ciphers. The hardware implementation of a stream cipher is also easier. When we need to encrypt binary streams and transmit them at a constant rate, a stream cipher is the better choice to use. Stream ciphers are also more immune to the corruption of bits during transmission.

The simplest and the most secure type of synchronous stream cipher is called the **one-time pad,** which was invented and patented by Gilbert Vernam. A one-time pad cipher uses a key stream that is randomly chosen for each encipherment. The encryption and decryption algorithms each use a single exclusive-or operation. Based on properties of the exclusive-or operation, the encryption and decryption algorithms are inverses of each other. It is important to note that in this cipher the exclusive-or operation is used one bit at a time. Note also that there must be a secure channel so that Alice can send the key stream sequence to Bob (Figure 29.12).

**Figure 29.12** *One-time pad*




The one-time pad is an ideal cipher. It is perfect. There is no way that an adversary can guess the key or the plaintext and ciphertext statistics. There is no relationship between the plaintext and ciphertext, either. In other words, the ciphertext is a true

random stream of bits even if the plaintext contains some patterns. Eve cannot break the cipher unless she tries all possible random key streams, which would be $2^n$ if the size of the plaintext is $n$ bits. However, there is an issue here. How can the sender and the receiver share a one-time pad key each time they want to communicate? They need to somehow agree on the random key. So this perfect and ideal cipher is very difficult to achieve. However, there are some feasible, less secured, versions. One of the common alternatives is called a feedback shift register (FSR), but we leave the discussion of this interesting cipher to the books dedicated to the security topic.

## 29.4 ASYMMETRIC-KEY CIPHERS

In previous sections we discussed symmetric-key ciphers. In this chapter, we start the discussion of **asymmetric-key ciphers.** Symmetric- and asymmetric-key ciphers will exist in parallel and continue to serve the community. We actually believe that they are complements of each other; the advantages of one can compensate for the disadvantages of the other.

The conceptual differences between the two systems are based on how these systems keep a secret. In symmetric-key cryptography, the secret must be shared between two persons. In asymmetric-key cryptography, the secret is personal (unshared); each person creates and keeps his or her own secret.

In a community of $n$ people, $n(n - 1)/2$ shared secrets are needed for symmetric-key cryptography; only $n$ personal secrets are needed in asymmetric-key cryptography. For a community with a population of 1 million, symmetric-key cryptography would require half a billion shared secrets; asymmetric-key cryptography would require 1 million personal secrets.

**Symmetric-key cryptography is based on sharing secrecy;**
**asymmetric-key cryptography is based on personal secrecy.**

There are some other aspects of security besides encipherment that need asymmetric-key cryptography. These include authentication and digital signatures. Whenever an application is based on a personal secret, we need to use asymmetric-key cryptography.

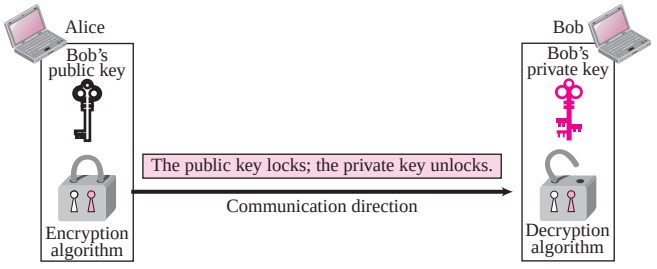
Whereas symmetric-key cryptography is based on substitution and permutation of symbols (characters or bits), asymmetric-key cryptography is based on applying mathematical functions to numbers. In symmetric-key cryptography, the plaintext and ciphertext are thought of as a combination of symbols. Encryption and decryption permute these symbols or substitute a symbol for another. In asymmetric-key cryptography, the plaintext and ciphertext are numbers; encryption and decryption are mathematical functions that are applied to numbers to create other numbers.

**In symmetric-key cryptography, symbols are permuted or substituted;**
**in asymmetric-key cryptography, numbers are manipulated.**

## Keys

Asymmetric key cryptography uses two separate keys: one private and one public. If encryption and decryption are thought of as locking and unlocking padlocks with keys, then the padlock that is locked with a public key can be unlocked only with the corresponding private key. Figure 29.13 shows that if Alice locks the padlock with Bob's public key, then only Bob's private key can unlock it.

**Figure 29.13** *Locking and unlocking in asymmetric-key cryptosystem*




The figure shows that, unlike symmetric-key cryptography, there are distinctive keys in asymmetric-key cryptography: a **private key** and a **public key.** Although some books use the term *secret key* instead of *private key,* we use the term *secret key* only for symmetric-key cryptography and the terms *private key* and *public key* for asymmetric-key cryptography. We even use different symbols to show the three keys. In other words, we want to show that a *secret key* is not exchangeable with a *private key;* there are two different types of secrets.

**Asymmetric-key ciphers are sometimes called public-key ciphers.**

## General Idea

Figure 29.14 shows the general idea of asymmetric-key cryptography as used for encipherment.

**Figure 29.14** *General idea of asymmetric-key cryptosystem*

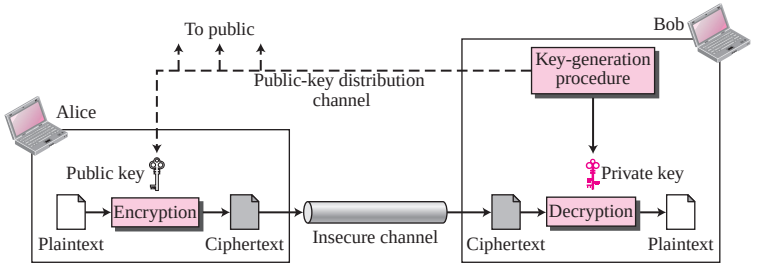


Figure 29.14 shows several important facts. First, it emphasizes the asymmetric nature of the cryptosystem. The burden of providing security is mostly on the shoulders of the receiver (Bob, in this case). Bob needs to create two keys: one private and one public. Bob is responsible for distributing the public key to the community. This can be done through a public-key distribution channel. Although this channel is not required to provide secrecy, it must provide authentication and integrity. Eve should not be able to advertise her public key to the community pretending that it is Bob's public key.

Second, asymmetric-key cryptography means that Bob and Alice cannot use the same set of keys for two-way communication. Each entity in the community should create its own private and public keys. Figure 29.14 shows how Alice can use Bob's public key to send encrypted messages to Bob. If Bob wants to respond, he needs to use Alice's public key.

Third, asymmetric-key cryptography means that Bob needs only one private key to receive all correspondence from anyone in the community, but Alice needs *n* public keys to communicate with *n* entities in the community, one public key for each entity. In other words, Alice needs a ring of public keys.

### *Plaintext/Ciphertext*

Unlike in symmetric-key cryptography, plaintext and ciphertext are treated as integers in asymmetric-key cryptography. The message must be encoded as an integer (or a set of integers) before encryption; the integer (or the set of integers) must be decoded into the message after decryption. Asymmetric-key cryptography is normally used to encrypt or decrypt small pieces of information, such as the cipher key for a symmetric-key cryptography. In other words, asymmetric-key cryptography normally is used for ancillary goals instead of message encipherment. However, these ancillary goals play a very important role in cryptography today.

### *Encryption/Decryption*

Encryption and decryption in asymmetric-key cryptography are mathematical functions applied over the numbers representing the plaintext and ciphertext. The ciphertext can be thought of as $C = \boldsymbol{f}(K_{public}, P)$; the plaintext can be thought of as $P = \boldsymbol{g}(K_{private}, C)$. The decryption function $\boldsymbol{f}$ is used only for encryption; the decryption function $\boldsymbol{g}$ is used only for decryption.

### *Need for Both*

There is a very important fact that is sometimes misunderstood: the advent of asymmetric-key (public-key) cryptography does not eliminate the need for symmetric-key (secret-key) cryptography. The reason is that asymmetric-key cryptography, which uses mathematical functions for encryption and decryption, is much slower than symmetric-key cryptography. For encipherment of large messages, symmetric-key cryptography is still needed. On the other hand, the speed of symmetric-key cryptography does not eliminate the need for asymmetric-key cryptography. Asymmetric-key cryptography is still needed for authentication, digital signatures, and secret-key exchanges. This means that, to be able to use all services of security today, we need both symmetric-key and asymmetric-key cryptography. One complements the other.

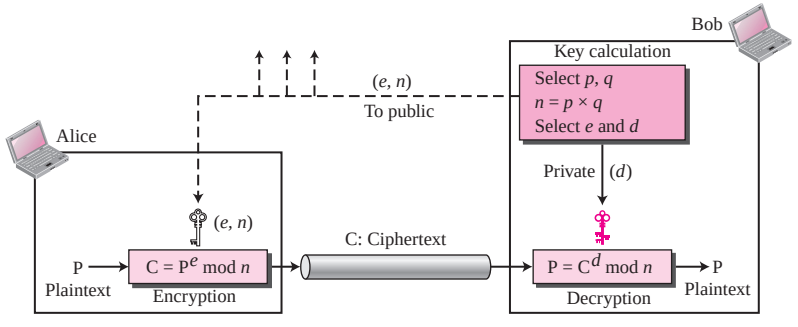
## RSA Cryptosystem

Although there are several asymmetric-key cryptosystems, one of the common public-key algorithms is the **RSA cryptosystem,** named for its inventors (Rivest, Shamir, and Adleman). RSA uses two exponents, $e$ and $d$, where $e$ is public and $d$ is private. Suppose P is the plaintext and C is the ciphertext. Alice uses $C = P^e \bmod n$ to create ciphertext C from plaintext P; Bob uses $P = C^d \bmod n$ to retrieve the plaintext sent by Alice. The modulus $n$, a very large number, is created during the key generation process.

### *Procedure*

Figure 29.15 shows the general idea behind the procedure used in RSA.

**Figure 29.15** *Encryption, decryption, and key generation in RSA*




Bob choose two large numbers, $p$ and $q$ and calculates $n = p \times q$ and $\phi = (p - 1) \times (q - 1)$. Bob then selects $e$ and $d$ such as $(e \times d) \bmod \phi = 1$. Bob advertises $e$ and $n$ to the community as the public key; Bob keeps $d$ as the secret key. Anyone, including Alice, can encrypt a message and send the ciphertext to Bob using $C = P^e$; Only Bob can decrypt the message using $P = C^d$. An intruder such as Eve cannot decrypt the message if $p$ and $q$ are very large numbers (she does not know $d$).

### Example 29.6

For the sake of demonstration, let Bob choose 7 and 11 as $p$ and $q$ and calculate $n = 7 \times 11 = 77$. The value of $\phi(n) = (7 - 1)(11 - 1)$, or 60. If he chooses $e$ to be 13, then $d$ is 37. Note that $e \times d \bmod 60 = 1$. Now imagine that Alice wants to send the plaintext 5 to Bob. She uses the public exponent 13 to encrypt 5. This system is not safe because $p$ and $q$ are small.

Plaintext: 5

$C = 5^{13} = 26 \bmod 77$

Ciphertext: 26

Ciphertext: 26

$P = 26^{37} = 5 \bmod 77$

Plaintext: 5

### Example 29.7

Here is a more realistic example calculated with a computer. We choose a 512-bit $p$ and $q$, calculate $n$ and $\phi(n)$, We then choose $e$ and calculate $d$. Finally, we show the results of encryption and decryption. The integer $p$ is a 159-digit number.

| | |
|---|---|
| $p$ = | 96130345313583504574191581280615427909309845594996215822583150879647940455056470638491257160180347503120986666064924201918087806674210960633542199266612 09 |

The integer $q$ is a 160-digit number.

| | |
|---|---|
| $q$ = | 12060191957231446918276794204450896001555925054637033936061798321731482148483764659215389453209175225273226830107120695604602513887145524969000359660045617 |

The modulus $n = p \times q$. It has 309 digits.

| | |
|---|---|
| $n$ = | 11593504173967614968892509864615887523771457375454144775485526137614788540832635081727687881596832516846884930062548576411125016241455233918292716250765772727460097082714127730434960500556347274566628060099924037102991424472292215772798531727033839381334692684137327622000966676671831831088373420823444370953 |

$\phi(n) = (p - 1)(q - 1)$ has 309 digits.

| | |
|---|---|
| $\phi(n)$ = | 11593504173967614968892509864615887523771457375454144775485526137614788540832635081727687881596832516846884930062548576411125016241455233918292716250765751054233608492916752034482627988117554787657013923444405716989581728196098226361075467211864612171359107358640614008885170265377277264467341066243857664128 |

Bob chooses $e$ = 35535 (the ideal is 65537). He then finds $d$.

| | |
|---|---|
| $e$ = | 35535 |
| $d$ = | 58008302860037763936093661289677917594669062089650962180422866111380593852822358731706286910030021710859044338402170729869087600611530620252495988444804756824096624708148581713046324064407770483313401085094738529564507193677406119732655742423721761761746207763716420760033708533328853214470885955136670294831 |

Alice wants to send the message "THIS IS A TEST", which can be changed to a numeric value using the 00–26 encoding scheme (26 is the *space* character).

**P =** 1907081826081826002619041819

The ciphertext calculated by Alice is $C = P^e$, which is

**C =**
475309123646226827206365550610545180942371796070491716523239243054
452960613199328566617843418359114151197411252005682979794571736036
101278218847892741566090480023507190715277185914975188465888632101
148354103361657898467968386763733765777465625079280521148141844048
141844308127730590046928742485591664621086 56

Bob can recover the plaintext from the ciphertext using $P = C^d$, which is

**P =** 1907081826081826002619041819

The recovered plaintext is "THIS IS A TEST" after decoding.

### Applications

Although RSA can be used to encrypt and decrypt actual messages, it is very slow if the message is long. RSA, therefore, is useful for short messages. In particular, we will see that RSA is used in digital signatures and other cryptosystems that often need to encrypt a small message without having access to a symmetric key. RSA is also used for authentication, as we will see later in the chapter.

## 29.5 MESSAGE INTEGRITY

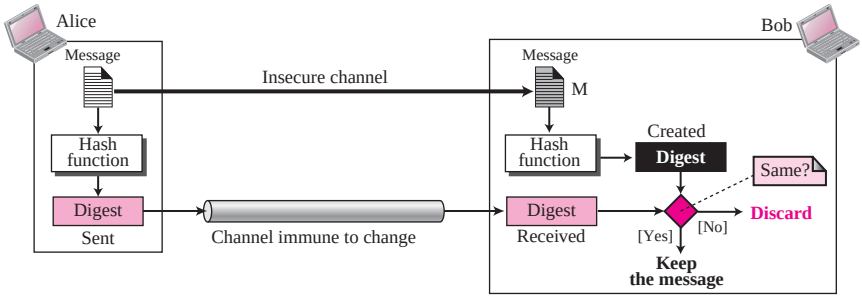
The cryptography systems that we have studied so far provide *secrecy*, or *confidentiality*, but not *integrity*. However, there are occasions where we may not even need secrecy but instead must have integrity. For example, Alice may write a will to distribute her estate upon her death. The will does not need to be encrypted. After her death, anyone can examine the will. The integrity of the will, however, needs to be preserved. Alice does not want the contents of the will to be changed.

### Message and Message Digest

One way to preserve the integrity of a document is through the use of a *fingerprint*. If Alice needs to be sure that the contents of her document will not be changed, she can put her fingerprint at the bottom of the document. Eve cannot modify the contents of this document or create a false document because she cannot forge Alice's fingerprint. To ensure that the document has not been changed, Alice's fingerprint on the document can be compared to Alice's fingerprint on file. If they are not the same, the document is not from Alice.The electronic equivalent of the document and fingerprint pair is the *message* and

*digest* pair. To preserve the integrity of a message, the message is passed through an algorithm called a **cryptographic hash function.** The function creates a compressed image of the message, called a **digest**, that can be used like a fingerprint. To check the integrity of a message, or document, Bob runs the cryptographic hash function again and compares the new digest with the previous one. If both are the same, Bob is sure that the original message has not been changed. Figure 29.16 shows the idea.

**Figure 29.16** *Message and digest*




The two pairs (document/fingerprint) and (message/message digest) are similar, with some differences. The document and fingerprint are physically linked together. The message and message digest can be unlinked (or sent separately), and, most importantly, the message digest needs to be safe from change.

**The message digest needs to be safe from change.**

## Hash Functions

A cryptographic hash function takes a message of arbitrary length and creates a message digest of fixed length. All cryptographic hash functions need to create a fixed-size digest out of a variable-size message. Creating such a function is best accomplished using iteration. Instead of using a hash function with variable-size input, a function with fixed-size input is created and is used a necessary number of times. The fixed-size input function is referred to as a **compression function.** It compresses an $n$-bit string to create an $m$-bit string where $n$ is normally greater than $m$. The scheme is referred to as an **iterated cryptographic hash function.**

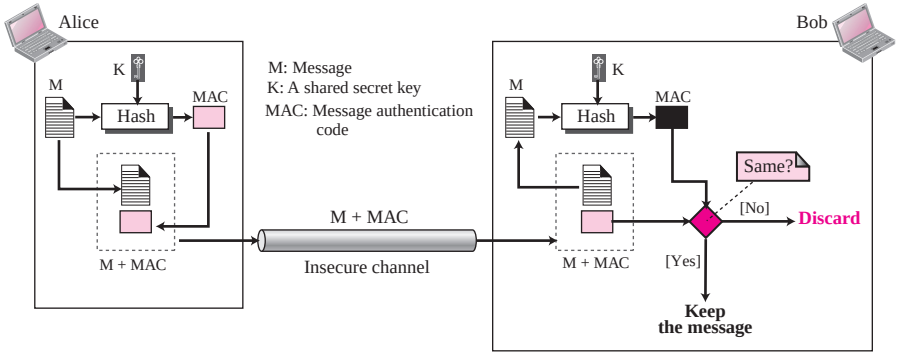
Several hash algorithms were designed by Ron Rivest. These are referred to as **MD2, MD4,** and **MD5,** where MD stands for Message Digest. The last version, MD5, is a strengthened version of MD4 that divides the message into blocks of 512 bits and creates a 128-bit digest. It turns out that a message digest of size 128 bits is too small to resist attack.

The **Secure Hash Algorithm (SHA)** is a standard that was developed by the National Institute of Standards and Technology (NIST). SHA has gone through several versions.

## 29.6 MESSAGE AUTHENTICATION

A digest can be used to check the integrity of a message: that the message has not been changed. To ensure the integrity of the message and the data origin authentication—that Alice is the originator of the message, not somebody else—we need to include a secret held by Alice (that Eve does not possess) in the process; we need to create a **message authentication code (MAC).** Figure 29.17 shows the idea.

**Figure 29.17** *Message authentication code*




Alice uses a hash function to create a MAC from the concatenation of the key and the message, h(K + M). She sends the message and the MAC to Bob over the insecure channel. Bob separates the message from the MAC. He then makes a new MAC from the concatenation of the message and the secret key. Bob then compares the newly created MAC with the one received. If the two MACs match, the message is authentic and has not been modified by an adversary.

Note that there is no need to use two channels in this case. Both message and the MAC can be sent on the same insecure channel. Eve can see the message, but she cannot forge a new message to replace it because Eve does not possess the secret key between Alice and Bob. She is unable to create the same MAC as Alice did.

**A MAC provides message integrity and message authentication using a combination of a hash function and a secret key.**

### HMAC

NIST has issued a standard for a nested MAC that is often referred to as **HMAC** (hashed MAC). The implementation of HMAC is much more complex than the simplified MAC.

## 29.7 DIGITAL SIGNATURE

Another way to provide message integrity and message authentication (and some more security services as we see shortly) is a digital signature. A MAC uses a secret key to protect the digest; a digital signature uses a pair of private-public keys.

**A digital signature uses a pair of private-public keys.**

We are all familiar with the concept of a signature. A person signs a document to show that it originated from her or was approved by her. The signature is proof to the recipient that the document comes from the correct entity. When a customer signs a check, the bank needs to be sure that the check is issued by that customer and nobody else. In other words, a signature on a document, when verified, is a sign of authentication—the document is authentic. Consider a painting signed by an artist. The signature on the art, if authentic, means that the painting is probably authentic.

When Alice sends a message to Bob, Bob needs to check the authenticity of the sender; he needs to be sure that the message comes from Alice and not Eve. Bob can ask Alice to sign the message electronically. In other words, an electronic signature can prove the authenticity of Alice as the sender of the message. We refer to this type of signature as a **digital signature.**

### Comparison

Let us begin by looking at the differences between conventional signatures and digital signatures.

#### *Inclusion*

A conventional signature is included in the document; it is part of the document. When we write a check, the signature is on the check; it is not a separate document. But when we sign a document digitally, we send the signature as a separate document.

#### *Verification Method*

The second difference between the two types of signatures is the method of verifying the signature. For a conventional signature, when the recipient receives a document, she compares the signature on the document with the signature on file. If they are the same, the document is authentic. The recipient needs to have a copy of this signature on file for comparison. For a digital signature, the recipient receives the message and the signature. A copy of the signature is not stored anywhere. The recipient needs to apply a verification technique to the combination of the message and the signature to verify the authenticity.

### Relationship

For a conventional signature, there is normally a one-to-many relationship between a signature and documents. A person uses the same signature to sign many documents. For a digital signature, there is a one-to-one relationship between a signature and a message. Each message has its own signature. The signature of one message cannot be used in another message. If Bob receives two messages, one after another, from Alice, he cannot use the signature of the first message to verify the second. Each message needs a new signature.

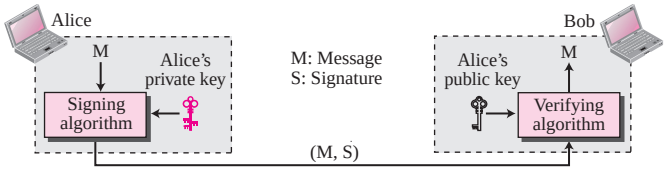
### Duplicity

Another difference between the two types of signatures is a quality called *duplicity.* With a conventional signature, a copy of the signed document can be distinguished from the original one on file. In digital signature, there is no such distinction unless there is a factor of time (such as a timestamp) on the document. For example, suppose Alice sends a document instructing Bob to pay Eve. If Eve intercepts the document and the signature, she can replay it later to get money again from Bob.

## Process

Figure 29.18 shows the digital signature process. The sender uses a **signing algorithm** to sign the message. The message and the signature are sent to the receiver. The receiver receives the message and the signature and applies the **verifying algorithm** to the combination. If the result is true, the message is accepted; otherwise, it is rejected.

**Figure 29.18** *Digital signature process*




A conventional signature is like a private "key" belonging to the signer of the document. The signer uses it to sign documents; no one else has this signature. The copy of the signature on file is like a public key; anyone can use it to verify a document, to compare it to the original signature.

In a digital signature, the signer uses her private key, applied to a signing algorithm, to sign the document. The verifier, on the other hand, uses the public key of the signer, applied to the verifying algorithm, to verify the document.

Note that when a document is signed, anyone, including Bob, can verify it because everyone has access to Alice's public key. Alice must not use her public key to sign the document because then anyone could forge her signature.

Can we use a secret (symmetric) key to both sign and verify a signature? The answer is negative for several reasons. First, a secret key is known by only two entities (Alice and Bob, for example). So if Alice needs to sign another document and send it to

Ted, she needs to use another secret key. Second, as we will see, creating a secret key for a session involves authentication, which uses a digital signature. We have a vicious cycle. Third, Bob could use the secret key between himself and Alice, sign a document, send it to Ted, and pretend that it came from Alice.

**A digital signature needs a public-key system.**
**The signer signs with her private key; the verifier verifies with the signer's public key.**

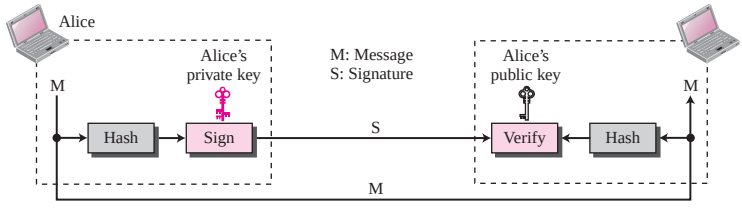
We should make a distinction between private and public keys as used in digital signatures and public and private keys as used in a cryptosystem for confidentiality. In the latter, the private and public keys of the receiver are used in the process. The sender uses the public key of the receiver to encrypt; the receiver uses his own private key to decrypt. In a digital signature, the private and public keys of the sender are used. The sender uses her private key; the receiver uses the sender's public key.

**A cryptosystem uses the private and public keys of the receiver:**
**a digital signature uses the private and public keys of the sender.**

## Signing the Digest

We said before that the asymmetric-key cryptosystems are very inefficient when dealing with long messages. In a digital signature system, the messages are normally long, but we have to use asymmetric-key schemes. The solution is to sign a digest of the message, which is much shorter than the message. A carefully selected message digest has a one-to-one relationship with the message. The sender can sign the message digest and the receiver can verify the message digest. The effect is the same. Figure 29.19 shows signing a digest in a digital signature system.

**Figure 29.19** *Signing the digest*




A digest is made out of the message at Alice's site. The digest then goes through the signing process using Alice's private key. Alice then sends the message and the signature to Bob.

At Bob's site, using the same public hash function, a digest is first created out of the received message. The verifying process is applied. If authentic, the message is accepted; otherwise, it is rejected.

## Services

We discussed several security services in the beginning of the chapter including *message confidentiality, message authentication, message integrity,* and *nonrepudiation*. A digital signature can directly provide the last three; for message confidentiality we still need encryption/decryption.

### *Message Authentication*

A secure digital signature scheme, like a secure conventional signature (one that cannot be easily copied) can provide message authentication (also referred to as data-origin authentication). Bob can verify that the message is sent by Alice because Alice's public key is used in verification. Alice's public key cannot verify the signature signed by Eve's private key.

### *Message Integrity*

The integrity of the message is preserved even if we sign the whole message because we cannot get the same signature if the message is changed. The digital signature schemes today use a hash function in the signing and verifying algorithms that preserves the integrity of the message.

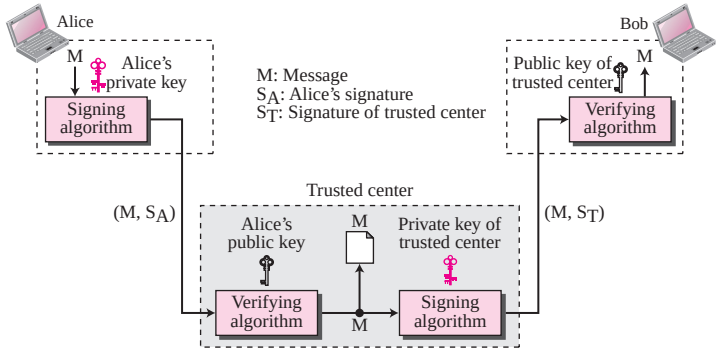
### *Nonrepudiation*

If Alice signs a message and then denies it, can Bob later prove that Alice actually signed it? For example, if Alice sends a message to a bank (Bob) and asks to transfer $10,000 from her account to Ted's account, can Alice later deny that she sent this message? With the scheme we have presented so far, Bob might have a problem. Bob must keep the signature on file and later use Alice's public key to create the original message to prove the message in the file and the newly created message are the same. This is not feasible because Alice may have changed her private or public key during this time; she may also claim that the file containing the signature is not authentic.

One solution is a trusted third party. People can create an established trusted party among themselves. Later in the chapter, we will see that a trusted party can solve many other problems concerning security services and key exchange. Figure 29.20 shows how a trusted party can prevent Alice from denying that she sent the message.

Alice creates a signature from her message ($S_A$) and sends the message, her identity, Bob's identity, and the signature to the center. The center, after checking that Alice's public key is valid, verifies through Alice's public key that the message came from Alice. The center then saves a copy of the message with the sender identity, recipient identity, and a timestamp in its archive. The center uses its private key to create another signature ($S_T$) from the message. The center then sends the message, the new signature, Alice's identity, and Bob's identity to Bob. Bob verifies the message using the public key of the trusted center.

**Figure 29.20** *Using a trusted center for nonrepudiation*




If in the future Alice denies that she sent the message, the center can show a copy of the saved message. If Bob's message is a duplicate of the message saved at the center, Alice will lose the dispute. To make everything confidential, a level of encryption/decryption can be added to the scheme, as discussed in the next section.

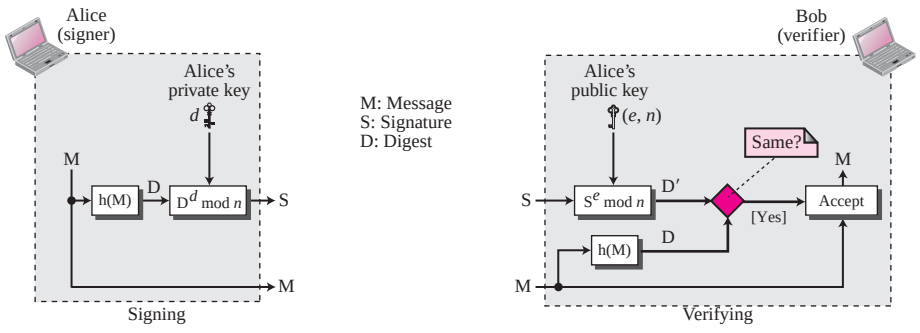
### *Confidentiality*

A digital signature does not provide confidential communication. If confidentiality is required, the message and the signature must be encrypted using either a secret-key or public-key cryptosystem.

## RSA Digital Signature Scheme

Several **digital signature schemes** have evolved during the last few decades. Some of them have been implemented. In this section, we briefly show one of them, RSA. In a previous section, we discussed how to use RSA cryptosystem to provide privacy. The RSA idea can also be used for signing and verifying a message. In this case, it is called the **RSA digital signature scheme.** The digital signature scheme changes the roles of the private and public keys. First, the private and public keys of the sender, not the receiver, are used. Second, the sender uses her own private key to sign the document; the receiver uses the sender's public key to verify it. If we compare the scheme with the conventional way of signing, we see that the private key plays the role of the sender's own signature and the sender's public key plays the role of the copy of the signature that is available to the public. Obviously Alice cannot use Bob's public key to sign the message because then any other person could do the same. The signing and verifying sites use the same function, but with different parameters. The verifier compares the message and the output of the function for equality in modulo arithmetic. If the result is true, the message is accepted. Figure 29.21 shows the scheme in which the signing and verifying is done on the digest of the message instead of the message itself because the public-key cryptography is not very efficient to be used with long messages; the digest is much smaller than the message itself.

**Figure 29.21** *The RSA signature on the message digest*




Alice, the signer, first uses an agreed-upon hash function to create a digest from the message, D = h(M). She then signs the digest, $S = D^d \bmod n$. The message and the signature are sent to Bob. Bob, the verifier, receives the message and the signature. He first uses Alice's public exponent to retrieve the digest, $D' = S^e \bmod n$. He then applies the hash algorithm to the message received to obtain D = h(M). Bob now compares the two digests, D and D′. If they are equal (in modulo arithmetic), he accepts the message.

### Digital Signature Standard (DSS)

The **Digital Signature Standard (DSS)** was adopted by the National Institute of Standards and Technology (NIST) in 1994. DSS is a complicated, and more secure, digital signature scheme.

## 29.8 ENTITY AUTHENTICATION

Entity authentication is a technique designed to let one party prove the identity of another party. An *entity* can be a person, a process, a client, or a server. The entity whose identity needs to be proven is called the *claimant;* the party that tries to prove the identity of the claimant is called the *verifier.*

### Entity versus Message Authentication

There are two differences between *entity authentication and message authentication (data-origin authentication).*

1. Message authentication (or data-origin authentication) might not happen in real time; entity authentication does. In the former, Alice sends a message to Bob. When Bob authenticates the message, Alice may or may not be present in the communication process. On the other hand, when Alice requests entity authentication, there is no real message communication involved until Alice is authenticated by Bob. Alice needs to be online and to take part in the process. Only after she is authenticated can messages be communicated between Alice and Bob. Data-origin authentication is required when an e-mail is sent from Alice

to Bob. Entity authentication is required when Alice gets cash from an automatic teller machine.

2. Second, message authentication simply authenticates one message; the process needs to be repeated for each new message. Entity authentication authenticates the claimant for the entire duration of a session.

## Verification Categories

In entity authentication, the claimant must identify herself to the verifier. This can be done with one of three kinds of witnesses: *something known*, *something possessed*, or *something inherent*.

- **Something known.** This is a secret known only by the claimant that can be checked by the verifier. Examples are a password, a PIN, a secret key, and a private key.
- **Something possessed.** This is something that can prove the claimant's identity. Examples are a passport, a driver's license, an identification card, a credit card, and a smart card.
- **Something inherent.** This is an inherent characteristic of the claimant. Examples are conventional signatures, fingerprints, voice, facial characteristics, retinal pattern, and handwriting.

In this section, we only discuss the first type of witness, *something known*, which is normally used for remote (online) entity authentication. The other two categories are normally used when the claimant is personally present.

## Passwords

The simplest and oldest method of entity authentication is the use of a **password**, which is something that the claimant *knows*. A password is used when a user needs to access a system for using the system's resources (login). Each user has a user identification that is public, and a password that is private. Passwords, however, are very prone to attack. A password can be stolen, intercepted, guessed, and so on.

## Challenge-Response

In password authentication, the claimant proves her identity by demonstrating that she knows a secret, the password. However, because the claimant sends this secret, it is susceptible to interception by the adversary. In **challenge-response authentication,** the claimant proves that she *knows* a secret without sending it. In other words, the claimant does not send the secret to the verifier; the verifier either has it or finds it.

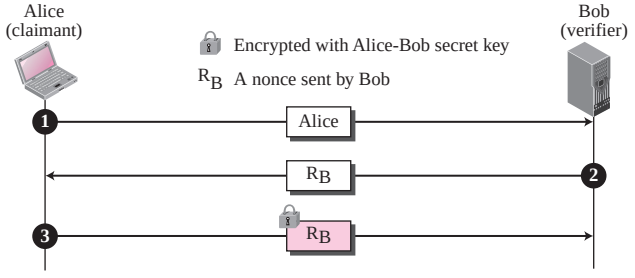
**In challenge-response authentication, the claimant proves that she knows a secret without sending it to the verifier.**

The *challenge* is a time-varying value such as a random number or a timestamp that is sent by the verifier. The claimant applies a function to the challenge and sends the result, called a *response*, to the verifier. The response shows that the claimant knows the secret.

### Using a Symmetric-Key Cipher

Several approaches to challenge-response authentication use symmetric-key encryption. The secret here is the shared secret key, known by both the claimant and the verifier. The function is the encrypting algorithm applied on the challenge. Although there are several approaches to this method, we just show the simplest one to give an idea. Figure 29.22 shows this first approach.

**Figure 29.22** *Unidirectional, symmetric-key authentication*



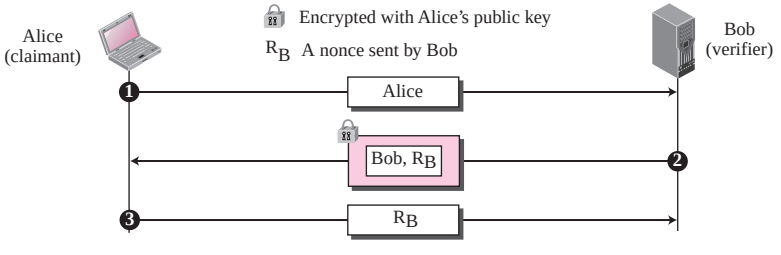

The first message is not part of challenge-response, it only informs the verifier that the claimant wants to be challenged. The second message is the challenge. $R_B$ is the nonce (abbreviation for *number once*) randomly chosen by the verifier (Bob) to challenge the claimant. The claimant encrypts the nonce using the shared secret key known only to the claimant and the verifier and sends the result to the verifier. The verifier decrypts the message. If the nonce obtained from decryption is the same as the one sent by the verifier, Alice is authenticated.

Note that in this process, the claimant and the verifier need to keep the symmetric key used in the process secret. The verifier must also keep the value of the nonce for claimant identification until the response is returned.

### Using an Asymmetric-Key Cipher

Figure 29.23 shows this approach.

**Figure 29.23** *Unidirectional, asymmetric-key authentication*

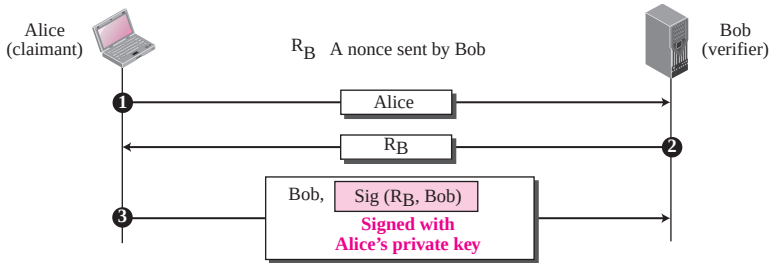


Instead of a symmetric-key cipher, we can use an asymmetric-key cipher for entity authentication. Here the secret must be the private key of the claimant. The claimant must show that she owns the private key related to the public key that is available to everyone. This means that the verifier must encrypt the challenge using the public key of the claimant; the claimant then decrypts the message using her private key. The response to the challenge is the decrypted challenge. If the $R_B$ received in the third message is the same sent in the second message, Alice is authenticated.

### *Using Digital Signature*

Entity authentication can also be achieved using a digital signature. When a digital signature is used for entity authentication, the claimant uses her private key for signing. In the first approach, shown in Figure 29.24, Bob uses a plaintext challenge and Alice signs the response. If the $R_B$ received in the third message is the same sent in the second message, Alice is authenticated.

**Figure 29.24** *Digital signature, unidirectional authentication*




## 29.9 KEY MANAGEMENT

We discussed symmetric-key and asymmetric-key cryptography in the previous sections. However, we have not yet discussed how secret keys in symmetric-key cryptography, and public keys in asymmetric-key cryptography, are distributed and maintained. This section touches on these two issues.

### Symmetric-Key Distribution

Symmetric-key cryptography is more efficient than asymmetric-key cryptography for enciphering large messages. Symmetric-key cryptography, however, needs a shared secret key between two parties.

If Alice needs to exchange confidential messages with $N$ people, she needs $N$ different keys. What if $N$ people need to communicate with each other? A total of $N(N - 1)$ keys is needed if we require that Alice and Bob use two keys for bidirectional communication; only $N(N - 1)/2$ keys are needed if we allow a key to be used for both directions. This means that if one million people need to communicate with each other, each person has almost one million different keys; in total, almost one billion keys are needed. This is normally referred to as the $N^2$ problem because the number of required keys for $N$ entities is close to $N^2$.

The number of keys is not the only problem; the distribution of keys is another. If Alice and Bob want to communicate, they need a way to exchange a secret key; if Alice wants to communicate with one million people, how can she exchange one million keys with one million people? Using the Internet is definitely not a secure method. It is obvious that we need an efficient way to maintain and distribute secret keys.

### *Key-Distribution Center: KDC*

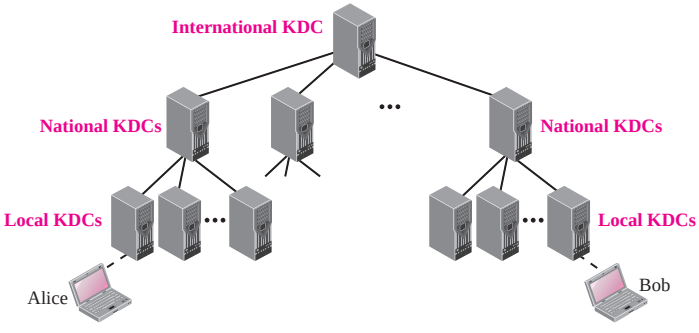
A practical solution is the use of a trusted third party, referred to as a **key-distribution center (KDC).** To reduce the number of keys, each person establishes a shared secret key with the KDC. A secret key is established between the KDC and each member. Now the question is how Alice can send a confidential message to Bob. The process is as follows:

1. Alice sends a request to the KDC stating that she needs a session (temporary) secret key between herself and Bob.
2. The KDC informs Bob about Alice's request.
3. If Bob agrees, a session key is created between the two.

The secret key between Alice and Bob that is established with the KDC is used to authenticate Alice and Bob to the KDC and to prevent Eve from impersonating either of them.

**Multiple KDCs** When the number of people using a KDC increases, the system becomes unmanageable and a bottleneck can result. To solve the problem, we need to have multiple KDCs. We can divide the world into domains. Each domain can have one or more KDCs (for redundancy in case of failure). Now if Alice wants to send a confidential message to Bob, who belongs to another domain, Alice contacts her KDC, which in turn contacts the KDC in Bob's domain. The two KDCs can create a secret key between Alice and Bob. There can be local KDCs, national KDCs, and international KDCs. When Alice needs to communicate with Bob, who lives in another country, she sends her request to a local KDC; the local KDC relays the request to the national KDC; the national KDC relays the request to an international KDC. The request is then relayed all the way down to the local KDC where Bob lives. Figure 29.25 shows a configuration of hierarchical multiple KDCs.

**Figure 29.25** *Multiple KDCs*

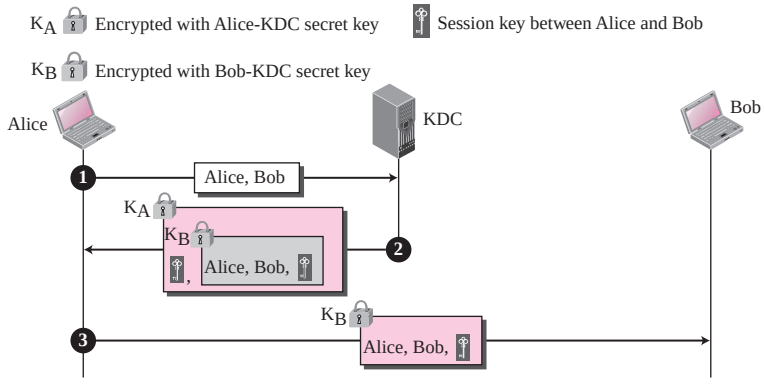


**Session Keys** A KDC creates a secret key for each member. This secret key can be used only between the member and the KDC, not between two members. If Alice needs to communicate secretly with Bob, she needs a secret key between herself and Bob. A KDC can create a **session key** between Alice and Bob, using their keys with the center. The keys of Alice and Bob are used to authenticate Alice and Bob to the center and to each other before the session key is established. After communication is terminated, the session key is no longer useful.

**A session symmetric key between two parties is used only once.**

Several different approaches have been proposed to create the session key using ideas discussed in the previous sections. We show the simplest approach in Figure 29.26. Although this approach is very rudimentary, it helps to understand more sophisticated approaches in the literature.

**Figure 29.26** *Creating a session key using KDC*




1. Alice sends a plaintext message to the KDC to obtain a symmetric session key between Bob and herself. The message contains her registered identity (the word *Alice* in the figure) and the identity of Bob (the word *Bob* in the figure). This message is not encrypted, it is public.The KDC does not care.
2. The KDC receives the message and creates what is called a **ticket.** The ticket is encrypted using Bob's key ($K_B$). The ticket contains the identities of Alice and Bob and the session key ($K_{AB}$). The ticket with a copy of the session key is sent to Alice. Alice receives the message, decrypts it, and extracts the session key. She cannot decrypt Bob's ticket; the ticket is for Bob, not for Alice. Note that this message contains a double encryption; the ticket is encrypted, and the entire message is also encrypted. In the second message, Alice is actually authenticated to the KDC, because only Alice can open the whole message using her secret key with KDC.

3. Alice sends the ticket to Bob. Bob opens the ticket and knows that Alice needs to send messages to him using $K_{AB}$ as the session key. Note that in this message, Bob is authenticated to the KDC because only Bob can open the ticket. Because Bob is authenticated to the KDC, he is also authenticated to Alice, who trusts the KDC. In the same way, Alice is also authenticated to Bob, because Bob trusts the KDC and the KDC has sent Bob the ticket that includes the identity of Alice.

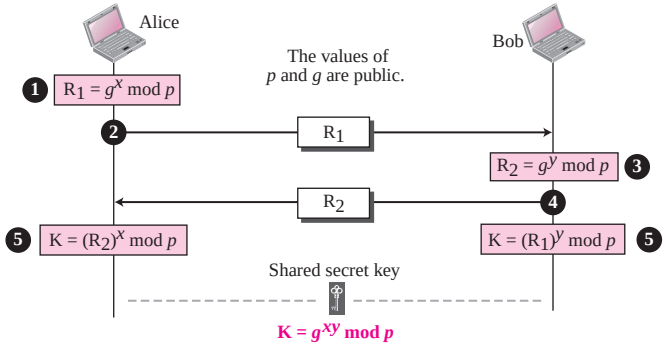
## Symmetric-Key Agreement

Alice and Bob can create a session key between themselves without using a KDC. This method of session-key creation is referred to as the symmetric-key agreement. Although there are several ways to accomplish this, we discuss only one method, Diffie-Hellman, which shows the basic idea used in more sophisticated (less prone to attacks) methods.

### *Diffie-Hellman Key Agreement*

In the **Diffie-Hellman protocol** two parties create a symmetric session key without the need of a KDC. Before establishing a symmetric key, the two parties need to choose two numbers $p$ and $g$. These two numbers have some properties discussed in number theory, but beyond the scope of this book. These two numbers do not need to be confidential. They can be sent through the Internet; they can be public. Figure 29.27 shows the procedure.

**Figure 29.27** *Diffie-Hellman method*




The steps are as follows:

1. Alice chooses a large random number $x$ such that $0 \le x \le p - 1$ and calculates $R_1 = g^x \bmod p$.
2. Alice sends $R_1$ to Bob. Note that Alice does not send the value of $x$; she sends only $R_1$.
3. Bob chooses another large random number $y$ such that $0 \le y \le p - 1$ and calculates $R_2 = g^y \bmod p$.
4. Bob sends $R_2$ to Alice. Again, note that Bob does not send the value of $y$, he sends only $R_2$.

5. Alice calculates $K = (R_2)^x \bmod p$. Bob also calculates $K = (R_1)^y \bmod p$.

K is the symmetric key for the session.

$$K = (g^x \bmod p)^y \bmod p = (g^y \bmod p)^x \bmod p = g^{xy} \bmod p$$

Bob has calculated $K = (R_1)^y \bmod p = (g^x \bmod p)^y \bmod p = g^{xy} \bmod p$. Alice has calculated $K = (R_2)^x \bmod p = (g^y \bmod p)^x \bmod = g^{xy} \bmod p$. Both have reached the same value without Bob knowing the value of $x$ and without Alice knowing the value of $y$.

**The symmetric (shared) key in the Diffie-Hellman method is $K = g^{xy} \bmod p$.**

### Example 29.8

Let us give a trivial example to make the procedure clear. Our example uses small numbers, but note that in a real situation, the numbers are very large. Assume that $g = 7$ and $p = 23$. The steps are as follows:

1. Alice chooses $x = 3$ and calculates $R_1 = 7^3 \bmod 23 = 21$.
2. Alice sends the number 21 to Bob.
3. Bob chooses $y = 6$ and calculates $R_2 = 7^6 \bmod 23 = 4$.
4. Bob sends the number 4 to Alice.
5. Alice calculates the symmetric key $K = 4^3 \bmod 23 = 18$. Bob calculates the symmetric key $K = 21^6 \bmod 23 = 18$.

   The value of K is the same for both Alice and Bob; $g^{xy} \bmod p = 7^{18} \bmod 35 = 18$.

## Public-Key Distribution

In asymmetric-key cryptography, people do not need to know a symmetric shared key. If Alice wants to send a message to Bob, she only needs to know Bob's public key, which is open to the public and available to everyone. If Bob needs to send a message to Alice, he only needs to know Alice's public key, which is also known to everyone. In public-key cryptography, everyone shields a private key and advertises a public key.

**In public-key cryptography, everyone has access to everyone's public key; public keys are available to the public.**

Public keys, like secret keys, need to be distributed to be useful. Let us briefly discuss the way public keys can be distributed.

### *Public Announcement*

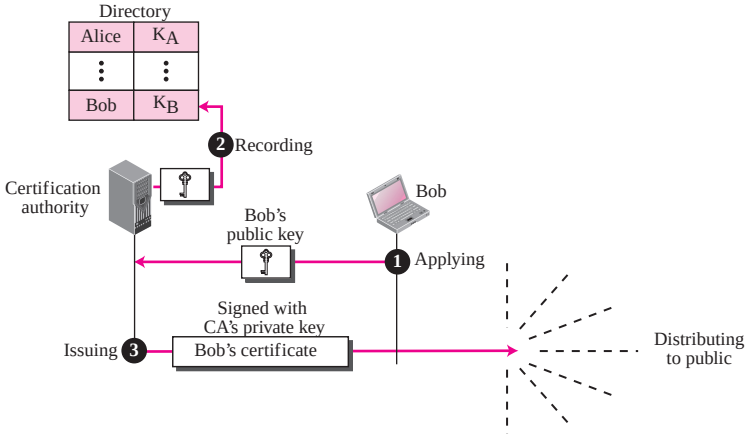
The naive approach is to announce public keys publicly. Bob can put his public key on his website or announce it in a local or national newspaper. When Alice needs to send a confidential message to Bob, she can obtain Bob's public key from his site or from the newspaper, or even send a message to ask for it. This approach, however, is not secure;

it is subject to forgery. For example, Eve could make such a public announcement. Before Bob can react, damage could be done. Eve can fool Alice into sending her a message that is intended for Bob. Eve could also sign a document with a corresponding forged private key and make everyone believe it was signed by Bob. The approach is also vulnerable if Alice directly requests Bob's public key. Eve can intercept Bob's response and substitute her own forged public key for Bob's public key.

### *Certification Authority*

The common approach to distributing public key is to create **public-key certificates.** Bob wants two things; he wants people to know his public key, and he wants no one to accept a forged public key as his. Bob can go to a **certification authority (CA),** a federal or state organization that binds a public key to an entity and issues a certificate. Figure 29.28 shows the concept.

**Figure 29.28** *Certification authority*




The CA has a well-known public key itself that cannot be forged. The CA checks Bob's identification (using a picture ID along with other proof). It then asks for Bob's public key and writes it on the certificate. To prevent the certificate itself from being forged, the CA signs the certificate with its private key. Now Bob can upload the signed certificate. Anyone who wants Bob's public key downloads the signed certificate and uses the center's public key to extract Bob's public key.

### *X.509*

Although the use of a CA has solved the problem of public-key fraud, it has created a side effect. Each certificate may have a different format. If Alice wants to use a program to automatically download different certificates and digests belonging to different people, the program may not be able to do this. One certificate may have the public key in one format and another in a different format. The public key may be on the first line in one certificate, and on the third line in another. Anything that needs to be used universally must have a universal format. To remove this side effect, the ITU has designed **X.509,** a

recommendation that has been accepted by the Internet with some changes. X.509 is a way to describe the certificate in a structured way. It uses a well-known protocol called ASN.1 (Abstract Syntax Notation 1) that defines fields familiar to computer programmers.

## 29.10 FURTHER READING

Several books give through coverage of cryptography and network security. In particular, we recommend [For 08], [Sta 06], [Bis 05], [Mao 04], and [Sti 06].

## 29.11 KEY TERMS

additive cipher
asymmetric-key cipher
attacks
autokey cipher
availability
bit-oriented ciphers
block cipher
Caesar cipher
certification authority (CA)
challenge-response authentication
character-oriented ciphers
cipher
ciphertext
circular shift operation
combine operation
compression function
confidentiality
cryptography
cryptographic hash function
Data Encryption Standard (DES)
decryption
decryption algorithm
denial of service
Diffie-Hellman protocol
digest
digital signature
digital signature scheme
Digital Signature Standard (DSS)
encryption
encryption algorithm
entity authentication
HMAC
integrity
iterated cryptographic hash function
key
key-distribution center (KDC)
masquerading
MD2
MD4
MD5
message authentication code (MAC)
modern block cipher
modern stream cipher
modification
monoalphabetic cipher
one-time pad
password
P-box
plaintext
polyalphabetic cipher
polyalphabetic substitution
private key
public key
public-key certificate
replaying
repudiation
RSA cryptosystem
RSA digital signature scheme
S-box
Secure Hash Algorithm (SHA)
session key
shared secret key
shift cipher
signing algorithm
snooping
split operation
steganography
stream cipher
substitution cipher
swap operation
symmetric-key cipher
ticket
traffic analysis
transposition cipher
verifying algorithm
X.509

## 29.12 SUMMARY

- Information is an asset that has a value that needs to be secured; information needs to be hidden from unauthorized access (*confidentiality*), protected from unauthorized change (*integrity*), and available to an authorized entity when it is needed (*availability*). Our three goals of security can be threatened by security attacks. Two techniques have been devised to protect information against attacks: cryptography and steganography.
- Traditional ciphers are called symmetric-key ciphers because the same key is used for encryption and decryption, and the key can be used for bidirectional communication. We can divide traditional symmetric-key ciphers into two broad categories: substitution ciphers and transposition ciphers. A substitution cipher substitutes a character by another; a transposition cipher reorders the characters.
- Modern ciphers are bit-oriented ciphers. A modern cipher can be either a block cipher or a stream cipher. A modern block cipher uses several rounds of a combination of substitution, transposition, exclusive-or, and other elements to mix a block of bits. One of the common block ciphers used today is DES. A modern stream cipher encrypts and decrypts a stream of bits one bit at a time.
- Asymmetric key cryptography uses two separate keys: one private and one public. Asymmetric-key cryptography means that Bob and Alice cannot use the same set of keys for two-way communication. Bob needs only one private key to receive all correspondence from anyone in the community, but Alice needs *n* public keys to communicate with *n* entities in the community.
- One way to preserve the integrity of a document is through the use of a message digest. The message is passed through an algorithm called a cryptographic hash function. The function creates a compressed image of the message, called a digest, that can be used like a fingerprint.
- To ensure the integrity of the message and the data origin authentication we need to create a message authentication code (MAC), which includes the secret-key between Alice and Bob in the hash function. Another way to provide message integrity and message authentication is to use a digital signature. A MAC uses a secret key to protect the digest; a digital signature uses a pair of private-public keys to do so.
- Entity authentication is a technique designed to let one party prove the identity of another party. An *entity* can be a person, a process, a client, or a server. The entity whose identity needs to be proved is called the *claimant;* the party that tries to prove the identity of the claimant is called the *verifier.*
- To use symmetric-key and asymmetric-key cryptography, we need to manage keys. In symmetric-key cryptography, we can use the services of a KDC for creation of session keys between two entities. In asymmetric-key cryptography, we can use the service of a certification authority (CA) to issue certified public keys.

## 29.13 PRACTICE SET

### Exercises

1. Define the type of the attack in each of the following cases:
   - **a.** A student breaks into a professor's office to obtain a copy of the next test.
   - **b.** A student gives a check for $10 to buy a used book. Later the student finds out that the check was cashed for $100.
   - **c.** A student sends hundreds of e-mails per day to the school using a phony return e-mail address.
2. A small private club has only 100 members. Answer the following questions:
   - **a.** How many secret keys are needed if all members of the club need to send secret messages to each other?
   - **b.** How many secret keys are needed if everyone trusts the president of the club? If a member needs to send a message to another member, she first sends it to the president; the president then sends the message to the other member.
   - **c.** How many secret keys are needed if the president decides that the two members who need to communicate should contact him first. The president then creates a temporary key to be used between the two. The temporary key is encrypted and sent to both members.
3. Alice can use only the additive cipher on her computer to send a message to a friend. She thinks that the message is more secure if she encrypts the message two times, each time with a different key. Is she right? Defend your answer.
4. Encrypt the message "this is an exercise" using additive cipher with key = 20. Ignore the space between words. Decrypt the message to get the original plaintext.
5. **a.** Show the result of 3-bit circular left shift on the word $(10011011)_2$.
   - **b.** Show the result of 3-bit circular right shift on the word resulting from part a.
   - **c.** Compare the result of part b with the original word in part a.
6. **a.** Swap the word $(10011011)_2$.
   - **b.** Swap the word resulting from part a.
   - **c.** Compare the result of part a and part b to show that swapping is a self-invertible operation.
7. Find the result of the following operations:
   - **a.** $(01001101) \oplus (01001101)$
   - **b.** $(01001101) \oplus (10110010)$
8. The leftmost bit of a $4 \times 3$ S-box rotates the other three bits. If the leftmost bit is 0, the three other bits are rotated to the right 1 bit (circular right shift). If the leftmost bit is 1, the three other bits are rotated to the left one bit (circular left shift). If the input is 1011, what is the output? If the input is 0110, what is the output?
9. In RSA, given $n = 12091$, $e = 13$, and $d = 3653$ encrypt the message "THIS IS TOUGH" using the 00 to 26 encoding scheme. Decrypt the ciphertext to find the original message. Use 4-digit plaintext or ciphertext blocks.

10. In RSA, why can't Bob choose 1 as the public key *e*?
11. Explain why private-public keys cannot be used in creating a MAC.
12. Assume we have a very simple message digest. Our unrealistic message digest is just one number between 0 and 25. The digest is initially set to 0. The cryptographic hash function adds the current value of the digest to the value of the current character (between 0 and 25). Addition is in modulo 26. What is the value of the digest if the message is "HELLO"? Why is this digest not secure?
13. Modify Figure 29.22 so that both Alice and Bob can be authenticated to each other.
14. Modify Figure 29.23 so that both Alice and Bob can be authenticated to each other.
15. Modify Figure 29.24 so that both Alice and Bob can be authenticated to each other.

## Research Activities

16. Use the literature to find about the historical polyalphabetic cipher Vigenere.
17. One of the interesting traditional ciphers is the Hill cipher. Use the literature to find out about this cipher.
18. We often hear about Feistel and non-Feistel ciphers. Use the literature or the Internet to find out about the main difference between these two types of ciphers. To which category does DES belong?
19. Since the size of the cipher key in DES is small (only 56 bits), one uses triple DES block cipher today. Use the literature and find out how we can change DES to triple DES with only two keys. What is the size of the key in the triple DES?
20. Advanced Encryption Standard (AES) is a new modern block cipher. Use the literature to learn about it and compare and contrast it with DES. Is AES a Feistel or a non-Feistel cipher?
21. Another asymmetric-key cipher is called ElGamal. Use the literature to find out about it and compare it with RSA.
22. Use the literature to find the outline of the HMAC (similar to the one for MAC in Figure 29.17).
23. A very promising cryptographic hash function is Whirlpool. Use the literature and find some facts about this function. What is the relationship between this function and AES block cipher.
24. Use the literature to find out more about DSS.
25. A very interesting idea in using password in UNIX is to *salt* the password. Use the literature to find out about *salted* passwords.
26. Use the literature to find out about Lamport one-time passwords.
27. A hot issue in entity authentication today is the topic of *zero knowledge.* Use the literature to find out about this issue. How it is used in entity authentication?
28. A very common interesting KDC is a protocol referred to as Kerberos. Use the Internet to find out about it. Compare it with the simple KDS scheme we discussed in this chapter.
29. The key agreement protocol, Diffie-Hellman, we used in this chapter is not very secure. It is prone to the man-in-the-middle attack. Another protocol called station-to-station protocol has been defined to improve the security of Diffie-Hellman. Find out about this protocol and compare it with Diffie-Hellman.

CHAPTER

# 30

# *Internet Security*

The security techniques we discussed in Chapter 29 are combined to provide security services in the Internet at the network layer, transport layer, and application layer. The discussion and application of these techniques are very complicated, with several protocols involved and tens of different packets. Although the study of these protocols and details of these packets are very interesting, they need several hundred pages, if not thousands. In this last chapter of the book, we give a glance at these protocols to prepare the reader for more advanced books on this topic.

## OBJECTIVES

*The chapter has several objectives:*

- To introduce the idea of Internet security at the network layer and the IPSec protocol that implements that idea in two modes: transport and tunnel.
- To discuss two protocols in IPSec, AH and ESP, and explain the security services each provide.
- To introduce security association and its implementation in IPSec.
- To introduce virtual private networks (VPN) as an application of IPSec in the tunnel mode.
- To introduce the idea of Internet security at the transport layer and the SSL protocol that implements that idea.
- To show how SSL creates six cryptographic secrets to be used by the client and the server.
- To discuss four protocols used in SSL and how they are related to each other.
- To introduce Internet security at the application level and two protocols, PGP and S/MIME, that implement that idea.
- To show how PGP and S/MIME can provide confidentiality and message authentication.
- To discuss firewalls and their applications in protecting a site from intruders.

## 30.1 NETWORK LAYER SECURITY

We start this chapter with the discussion of security at the network layer. Although in the next two sections we discuss security at the transport and application layers, we also need security at the network layer for three reasons. First, not all client/server programs are protected at the application layer. Second, not all client/server programs at the application layer use the services of TCP to be protected by the transport layer security that we discuss for the transport layer; some programs use the service of UDP. Third, many applications, such as routing protocols, directly use the service of IP; they need security services at the IP layer.

**IP Security (IPSec)** is a collection of protocols designed by the Internet Engineering Task Force (IETF) to provide security for a packet at the network level. IPSec helps create authenticated and confidential packets for the IP layer.

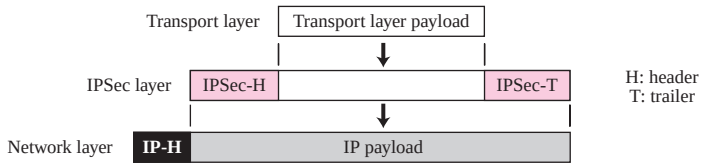
### Two Modes

IPSec operates in one of two different modes: transport mode or tunnel mode.

#### *Transport Mode*

In **transport mode,** IPSec protects what is delivered from the transport layer to the network layer. In other words, transport mode protects the payload to be encapsulated in the network layer, as shown in Figure 30.1.

**Figure 30.1** *IPSec in transport mode*

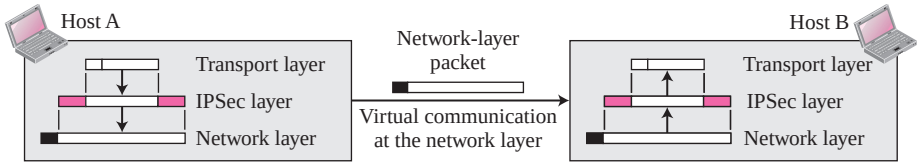



Note that transport mode does not protect the IP header. In other words, transport mode does not protect the whole IP packet; it protects only the packet from the transport layer (the IP layer payload). In this mode, the IPSec header (and trailer) are added to the information coming from the transport layer. The IP header is added later.

**IPSec in transport mode does not protect the IP header; it only protects the information coming from the transport layer.**

Transport mode is normally used when we need host-to-host (end-to-end) protection of data. The sending host uses IPSec to authenticate and/or encrypt the payload delivered from the transport layer. The receiving host uses IPSec to check the authentication and/or decrypt the IP packet and deliver it to the transport layer. Figure 30.2 shows this concept.

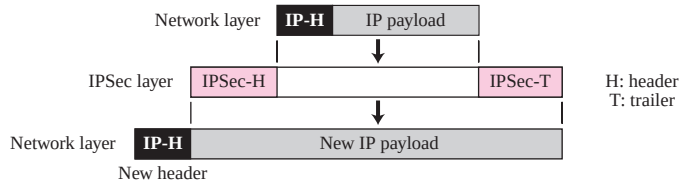
**Figure 30.2** *Transport mode in action*




### *Tunnel Mode*

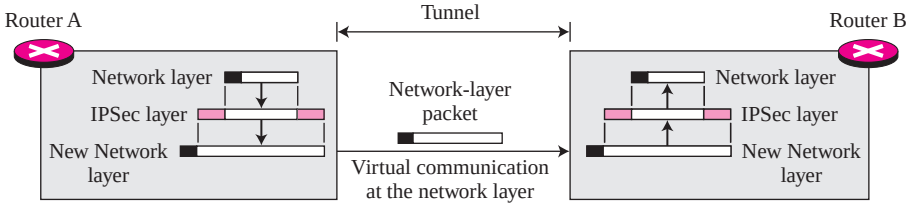
In **tunnel mode,** IPSec protects the entire IP packet. It takes an IP packet, including the header, applies IPSec security methods to the entire packet, and then adds a new IP header, as shown in Figure 30.3.

**Figure 30.3** *IPSec in tunnel mode*




The new IP header, as we will see shortly, has different information than the original IP header. Tunnel mode is normally used between two routers, between a host and a router, or between a router and a host, as shown in Figure 30.4. The entire original packet is protected from intrusion between the sender and the receiver, as if the whole packet goes through an imaginary tunnel.

**Figure 30.4** *Tunnel mode in action*



**IPSec in tunnel mode protects the original IP header.**

### *Comparison*

In transport mode, the IPSec layer comes between the transport layer and the network layer. In tunnel mode, the flow is from the network layer to the IPSec layer and then back to the network layer again. Figure 30.5 compares the two modes.

**Figure 30.5** *Transport mode versus tunnel mode*

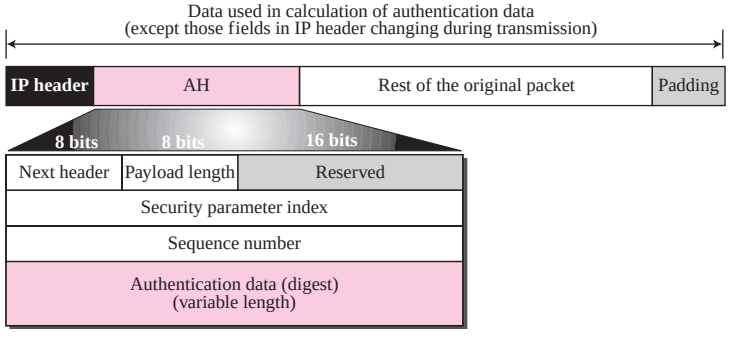

## Two Security Protocols

IPSec defines two protocols—the Authentication Header (AH) Protocol and the Encapsulating Security Payload (ESP) Protocol—to provide authentication and/or encryption for packets at the IP level.

### *Authentication Header (AH)*

The **Authentication Header (AH) Protocol** is designed to authenticate the source host and to ensure the integrity of the payload carried in the IP packet. The protocol uses a hash function and a symmetric (secret) key to create a message digest; the digest is inserted in the authentication header (see MAC in Chapter 29). The AH is then placed in the appropriate location, based on the mode (transport or tunnel). Figure 30.6 shows the fields and the position of the authentication header in transport mode.

**Figure 30.6** *Authentication Header (AH) protocol*



When an IP datagram carries an authentication header, the original value in the protocol field of the IP header is replaced by the value 51. A field inside the authentication header (the next header field) holds the original value of the protocol field (the type of payload being carried by the IP datagram). The addition of an authentication header follows these steps:

1. An authentication header is added to the payload with the authentication data field set to 0.
2. Padding may be added to make the total length even for a particular hashing algorithm.
3. Hashing is based on the total packet. However, only those fields of the IP header that do not change during transmission are included in the calculation of the message digest (authentication data).
4. The authentication data are inserted in the authentication header.
5. The IP header is added after changing the value of the protocol field to 51.

A brief description of each field follows:

- **Next header.** The 8-bit next header field defines the type of payload carried by the IP datagram (such as TCP, UDP, ICMP, or OSPF).
- **Payload length.** The name of this 8-bit field is misleading. It does not define the length of the payload; it defines the length of the authentication header in 4-byte multiples, but it does not include the first 8 bytes.
- **Security parameter index.** The 32-bit security parameter index (SPI) field plays the role of a virtual circuit identifier and is the same for all packets sent during a connection called a Security Association (discussed later).
- **Sequence number.** A 32-bit sequence number provides ordering information for a sequence of datagrams. The sequence numbers prevent a playback. Note that the sequence number is not repeated even if a packet is retransmitted. A sequence number does not wrap around after it reaches $2^{32}$; a new connection must be established.
- **Authentication data.** Finally, the authentication data field is the result of applying a hash function to the entire IP datagram except for the fields that are changed during transit (e.g., time-to-live).

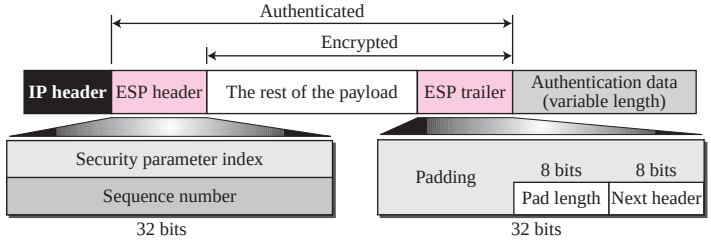
**The AH protocol provides source authentication and data integrity, but not privacy.**

### *Encapsulating Security Payload (ESP)*

The AH protocol does not provide confidentiality, only source authentication and data integrity. IPSec later defined an alternative protocol, **Encapsulating Security Payload (ESP),** that provides source authentication, integrity, and confidentiality. ESP adds a header and trailer. Note that ESP's authentication data are added at the end of the packet, which makes its calculation easier. Figure 30.7 shows the location of the ESP header and trailer.

When an IP datagram carries an ESP header and trailer, the value of the protocol field in the IP header is 50. A field inside the ESP trailer (the next-header field) holds

**Figure 30.7** *Encapsulating Security Payload (ESP)*




the original value of the protocol field (the type of payload being carried by the IP datagram, such as TCP or UDP). The ESP procedure follows these steps:

1. An ESP trailer is added to the payload.
2. The payload and the trailer are encrypted.
3. The ESP header is added.
4. The ESP header, payload, and ESP trailer are used to create the authentication data.
5. The authentication data are added to the end of the ESP trailer.
6. The IP header is added after changing the protocol value to 50.

The fields for the header and trailer are as follows:

- **Security parameter index.** The 32-bit security parameter index field is similar to the one defined for the AH protocol.
- **Sequence number.** The 32-bit sequence number field is similar to the one defined for the AH protocol.
- **Padding.** This variable-length field (0 to 255 bytes) of 0s serves as padding.
- **Pad length.** The 8-bit pad-length field defines the number of padding bytes. The value is between 0 and 255; the maximum value is rare.
- **Next header.** The 8-bit next-header field is similar to that defined in the AH protocol. It serves the same purpose as the protocol field in the IP header before encapsulation.
- **Authentication data.** Finally, the authentication data field is the result of applying an authentication scheme to parts of the datagram. Note the difference between the authentication data in AH and ESP. In AH, part of the IP header is included in the calculation of the authentication data; in ESP, it is not.

**ESP provides source authentication, data integrity, and privacy.**

### *IPv4 and IPv6*

IPSec supports both IPv4 and IPv6. In IPv6, however, AH and ESP are part of the extension header.

### AH versus ESP

The ESP protocol was designed after the AH protocol was already in use. ESP does whatever AH does with additional functionality (confidentiality). The question is, why do we need AH? The answer is that we don't. However, the implementation of AH is already included in some commercial products, which means that AH will remain part of the Internet until these products are phased out.

## Services Provided by IPSec

The two protocols, AH and ESP, can provide several security services for packets at the network layer. Table 30.1 shows the list of services available for each protocol.

**Table 30.1** *IPSec services*

| *Services* | *AH* | *ESP* |
|---|---|---|
| Access control | Yes | Yes |
| Message authentication (message integrity) | Yes | Yes |
| Entity authentication (data source authentication) | Yes | Yes |
| Confidentiality | No | Yes |
| Replay attack protection | Yes | Yes |

### Access Control

IPSec provides access control indirectly using a Security Association Database (SAD), as we will see in the next section. When a packet arrives at a destination, and there is no Security Association already established for this packet, the packet is discarded.

### Message Integrity

Message integrity is preserved in both AH and ESP. A digest of data is created and sent by the sender to be checked by the receiver.

### Entity Authentication

The Security Association and the keyed-hash digest of the data sent by the sender authenticate the sender of the data in both AH and ESP.

### Confidentiality

The encryption of the message in ESP provides confidentiality. AH, however, does not provide confidentiality. If confidentiality is needed, one should use ESP instead of AH.

### Replay Attack Protection

In both protocols, the replay attack is prevented by using sequence numbers and a sliding receiver window. Each IPSec header contains a unique sequence number when the Security Association is established. The number starts from 0 and increases until the value reaches $2^{32} - 1$. When the sequence number reaches the maximum, it is reset to 0 and, at the same time, the old Security Association (see the next section) is deleted and a new one is established. To prevent processing duplicate packets, IPSec mandates the use of a fixed-size window at the receiver. The size of the window is determined by the receiver with a default value of 64.

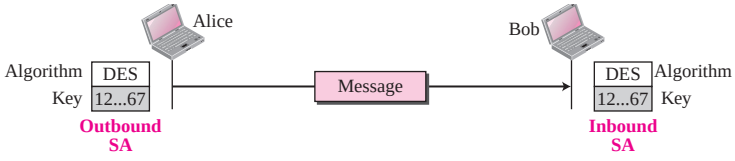
## Security Association

Security Association is a very important aspect of IPSec. IPSec requires a logical relationship, called a **Security Association (SA),** between two hosts. This section first discusses the idea and then shows how it is used in IPSec.

### *Idea of Security Association*

A Security Association is a contract between two parties; it creates a secure channel between them. Let us assume that Alice needs to unidirectionally communicate with Bob. If Alice and Bob are interested only in the confidentiality aspect of security, they can create a shared secret key between themselves. We can say that there are two Security Associations (SAs) between Alice and Bob; one outbound SA and one inbound SA. Each of them stores the value of the key in a variable and the name of the encryption/decryption algorithm in another. Alice uses the algorithm and the key to encrypt a message to Bob; Bob uses the algorithm and the key when he needs to decrypt the message received from Alice. Figure 30.8 shows a simple SA.

**Figure 30.8** *Simple SA*



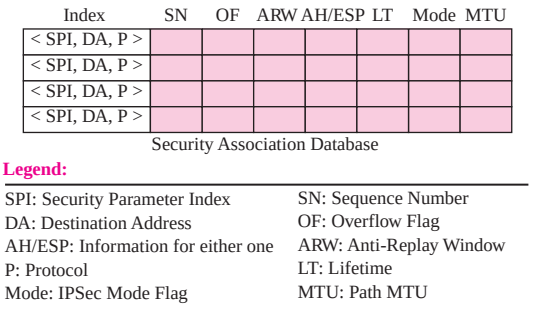

The Security Associations can be more involved if the two parties need message integrity and authentication. Each association needs other data such as the algorithm for message integrity, the key, and other parameters. It can be much more complex if the parties need to use specific algorithms and specific parameters for different protocols, such as IPSec AH or IPSec ESP.

### *Security Association Database (SAD)*

A Security Association can be very complex. This is particularly true if Alice wants to send messages to many people and Bob needs to receive messages from many people. In addition, each site needs to have both inbound and outbound SAs to allow bidirectional communication. In other words, we need a set of SAs that can be collected into a database. This database is called the **Security Association Database** (**SAD).** The database can be thought of as a two-dimensional table with each row defining a single SA. Normally, there are two SADs, one inbound and one outbound. Figure 30.9 shows the concept of outbound or inbound SADs for one entity.

When a host needs to send a packet that must carry an IPSec header, the host needs to find the corresponding entry in the outbound SAD to find the information for applying security to the packet. Similarly, when a host receives a packet that carries an IPSec header, the host needs to find the corresponding entry in the inbound SAD to find the information for checking the security of the packet. This searching must be specific in the sense that the receiving host needs to be sure that correct information is used for

**Figure 30.9** *SAD*




processing the packet. Each entry in an inbound SAD is selected using a triple index: security parameter index (a 32-bit number that defines the SA at the destination), destination address, and protocol (AH or ESP).

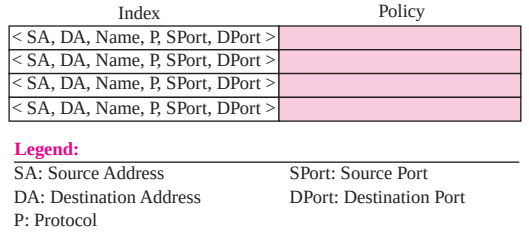
### *Security Policy*

Another important aspect of IPSec is the **Security Policy (SP),** which defines the type of security applied to a packet when it is to be sent or when it has arrived. Before using the SAD, discussed in the previous section, a host must determine the predefined policy for the packet.

### *Security Policy Database*

Each host that is using the IPSec protocol needs to keep a **Security Policy Database (SPD).** Again, there is a need for an inbound SPD and an outbound SPD. Each entry in the SPD can be accessed using a sextuple index: source address, destination address, name, protocol, source port, and destination port, as shown in Figure 30.10.

**Figure 30.10** *SPD*

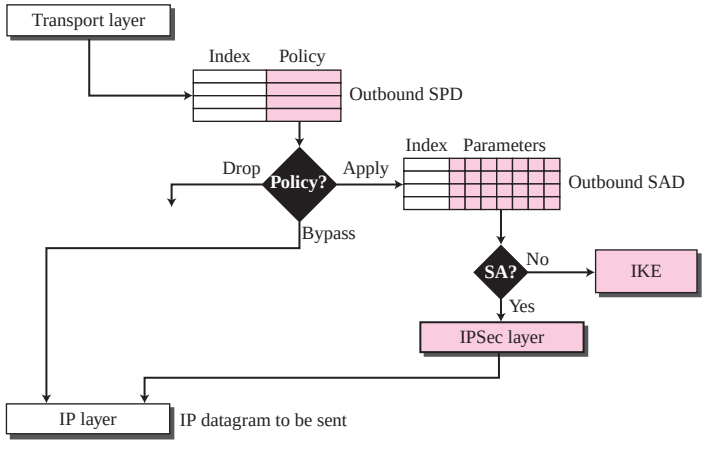



Source and destination addresses can be unicast, multicast, or wildcard addresses. The name usually defines a DNS entity. The protocol is either AH or ESP. The source and destination ports are the port addresses for the process running at the source and destination hosts.

**Outbound SPD** When a packet is to be sent out, the outbound SPD is consulted. Figure 30.11 shows the processing of a packet by a sender.

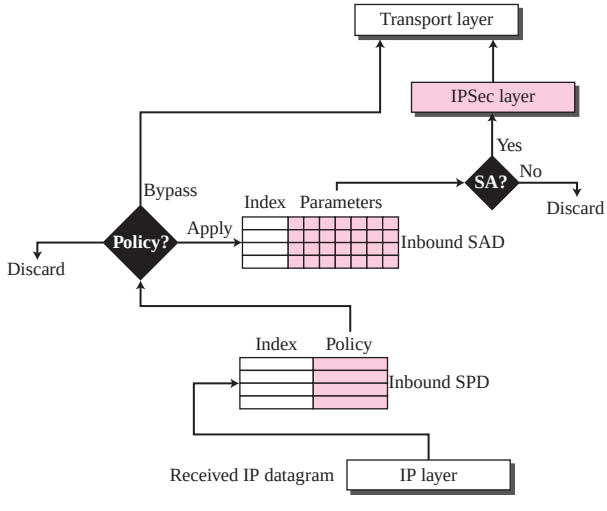
The input to the outbound SPD is the sextuple index; the output is one of the three following cases: drop (packet cannot be sent), bypass (bypassing security header), and apply (apply the security according to the SAD; if no SAD, create one).

**Figure 30.11** *Outbound processing*




**Inbound SPD** When a packet arrives, the inbound SPD is consulted. Each entry in the inbound SPD is also accessed using the same sextuple index. Figure 30.12 shows the processing of a packet by a receiver. The input to the inbound SPD is the sextuple index; the output is one of the three following cases: discard (drop the packet), bypass (bypass the security and deliver the packet to the transport layer), and apply (apply the policy using the SAD).

**Figure 30.12** *Inbound processing*



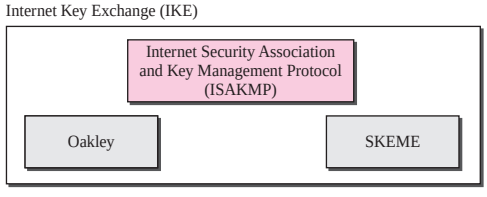
## Internet Key Exchange (IKE)

The **Internet Key Exchange (IKE)** is a protocol designed to create both inbound and outbound Security Associations. As we discussed in the previous section, when a peer needs to send an IP packet, it consults the Security Policy Database (SPD) to see if there is an SA for that type of traffic. If there is no SA, IKE is called to establish one.

**IKE creates SAs for IPSec.**

IKE is a complex protocol based on three other protocols: Oakley, SKEME, and ISAKMP, as shown in Figure 30.13.

**Figure 30.13** *IKE components*



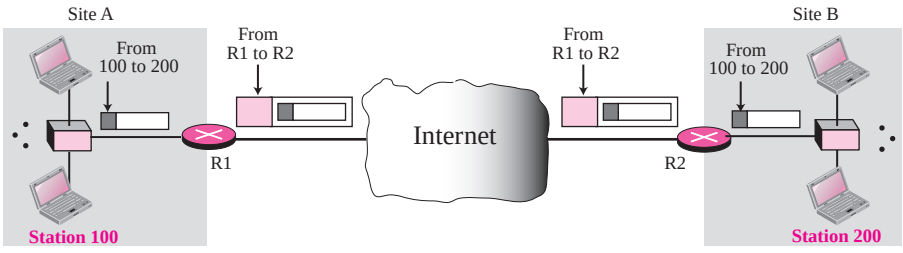

The **Oakley** protocol was developed by Hilarie Orman. It is a key creation protocol. **SKEME,** designed by Hugo Krawcyzk, is another protocol for key exchange. It uses public-key encryption for entity authentication in a key-exchange protocol.

The **Internet Security Association and Key Management Protocol (ISAKMP)** is a protocol designed by the National Security Agency (NSA) that actually implements the exchanges defined in IKE. It defines several packets, protocols, and parameters that allow the IKE exchanges to take place in standardized, formatted messages to create SAs. We leave the discussion of these three protocol for books dedicated to security.

## Virtual Private Network (VPN)

One of the applications of IPsec is in *virtual private networks*. A **virtual private network (VPN)** is a technology that is gaining popularity among large organizations that use the global Internet for both intra- and inter-organization communication, but require privacy in their intra-organization communication. VPN is a network that is private but virtual. It is private because it guarantees privacy inside the organization. It is virtual because it does not use real private WANs; the network is physically public but virtually private. Figure 30.14 shows the idea of a virtual private network. Routers R1 and R2 use VPN technology to guarantee privacy for the organization. VPN technology uses ESP protocol of IPSec in the tunnel mode. A private datagram, including the header, is encapsulated in an ESP packet. The router at the border of the sending site uses its own IP address and the address of the router at the destination site in the new datagram. The public network (Internet) is responsible for carrying the packet from R1 to R2. Outsiders cannot decipher the contents of the packet or the source and destination addresses. Deciphering takes place at R2, which finds the destination address of the packet and delivers it.

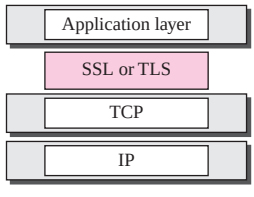
**Figure 30.14** *Virtual private network*




## 30.2 TRANSPORT LAYER SECURITY

Two protocols are dominant today for providing security at the transport layer: the **Secure Sockets Layer (SSL) protocol** and the **Transport Layer Security (TLS) protocol.** The latter is actually an IETF version of the former. We discuss SSL in this section; TLS is very similar. Figure 30.15 shows the position of SSL and TLS in the Internet model.

**Figure 30.15** *Location of SSL and TLS in the Internet model*




One of the goals of these protocols is to provide server and client authentication, data confidentiality, and data integrity. Application-layer client/server programs, such as HTTP (see Chapter 22), that use the services of TCP can encapsulate their data in SSL packets. If the server and client are capable of running SSL (or TLS) programs, then the client can use the URL *https://. . .* instead of *http://. . .* to allow HTTP messages to be encapsulated in SSL (or TLS) packets. For example, credit card numbers can be safely transferred via the Internet for online shoppers.

### SSL Architecture

SSL is designed to provide security and compression services to data generated from the application layer. Typically, SSL can receive data from any application layer protocol, but usually the protocol is HTTP. The data received from the application is compressed (optional), signed, and encrypted. The data is then passed to a reliable transport layer protocol such as TCP. Netscape developed SSL in 1994. Versions 2 and 3 were released in 1995. In this section, we discuss SSLv3.

### Services

SSL provides several services on data received from the application layer.

- **Fragmentation**. First, SSL divides the data into blocks of $2^{14}$ bytes or less.
- **Compression.** Each fragment of data is compressed using one of the lossless compression methods negotiated between the client and server. This service is optional.
- **Message Integrity.** To preserve the integrity of data, SSL uses a keyed-hash function to create a MAC (see Chapter 29).
- **Confidentiality.** To provide confidentiality, the original data and the MAC are encrypted using symmetric-key cryptography.
- **Framing**. A header is added to the encrypted payload. The payload is then passed to a reliable transport layer protocol.

### Key Exchange Algorithms

To exchange an authenticated and confidential message, the client and the server each need a set of cryptographic secrets. However, to create these secrets, one pre-master secret must be established between the two parties. SSL defines several key-exchange methods to establish this pre-master secret.

### Encryption/Decryption Algorithms

The client and server also need to agree to a set of encryption and decryption algorithms.

### Hash Algorithms

SSL uses hash algorithms to provide message integrity (message authentication). Several hash algorithms have also been defined for this purpose.

### Cipher Suite

The combination of key exchange, hash, and encryption algorithms defines a **cipher suite** for each SSL session.

### Compression Algorithms

Compression is optional in SSL. No specific compression algorithm is defined. Therefore a system can use whatever compression algorithm it desires.

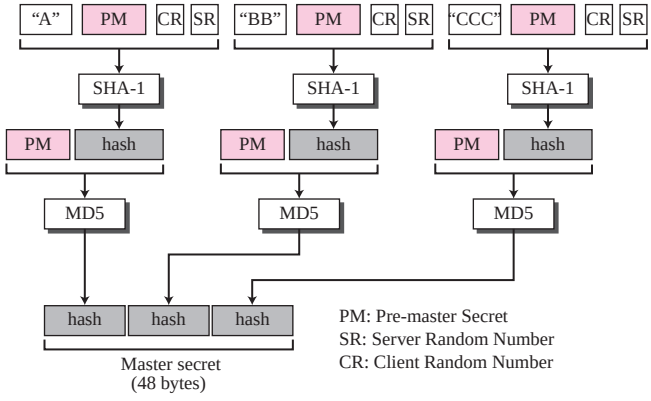
### Cryptographic Parameter Generation

To achieve message integrity and confidentiality, SSL needs six cryptographic secrets, four keys and two IVs (initialization vectors). The client needs one key for message authentication, one key for encryption, and one IV as original block in calculation. The server needs the same. SSL requires that the keys for one direction be different from those for the other direction. If there is an attack in one direction, the other direction is not affected. The parameters are generated using the following procedure:

1. The client and server exchange two random numbers; one is created by the client and the other by the server.

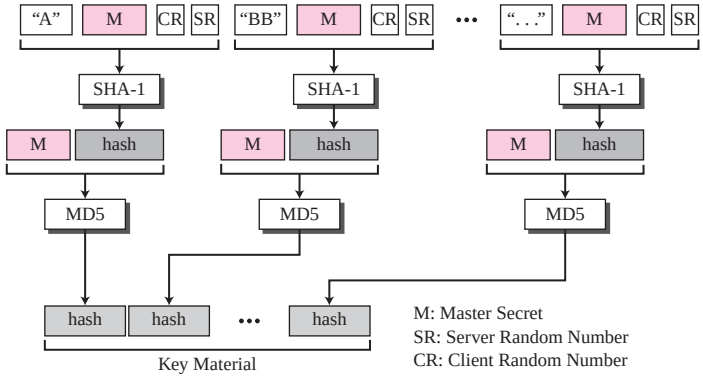
2. The client and server exchange one **pre-master secret** using one of the predefined key- exchange algorithms.
3. A 48-byte **master secret** is created from the pre-master secret by applying two hash functions (SHA-1 and MD5), as shown in Figure 30.16.

**Figure 30.16** *Calculation of master secret from pre-master secret*




4. The master secret is used to create variable-length **key material** by applying the same set of hash functions and prepending with different constants, as shown in Figure 30.17. The module is repeated until key material of adequate size is created.

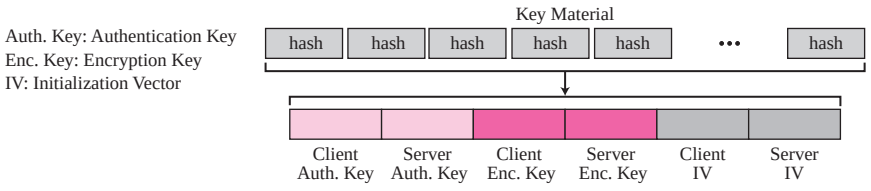
**Figure 30.17** *Calculation of key material from master secret*




Note that the length of the key material block depends on the cipher suite selected and the size of keys needed for this suite.

5. Six different secrets are extracted from the key material, as shown in Figure 30.18.

**Figure 30.18** *Extractions of cryptographic secrets from key material*




### *Sessions and Connections*

SSL differentiates a **connection** from a **session.** A session is an association between a client and a server. After a session is established, the two parties have common information such as the session identifier, the certificate authenticating each of them (if necessary), the compression method (if needed), the cipher suite, and a master secret that is used to create keys for message authentication encryption.

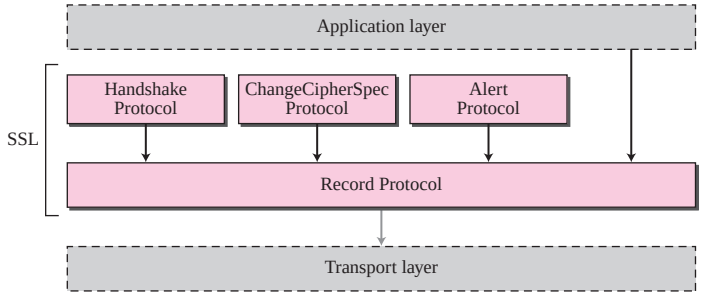
For two entities to exchange data, the establishment of a session is necessary, but not sufficient; they need to create a connection between themselves. The two entities exchange two random numbers and create, using the master secret, the keys and parameters needed for exchanging messages involving authentication and privacy.

A session can consist of many connections. A connection between two parties can be terminated and reestablished within the same session. When a connection is terminated, the two parties can also terminate the session, but it is not mandatory. A session can be suspended and resumed again.

## Four Protocols

We have discussed the idea of SSL without showing how SSL accomplishes its tasks. SSL defines four protocols in two layers, as shown in Figure 30.19.

**Figure 30.19** *Four SSL protocols*



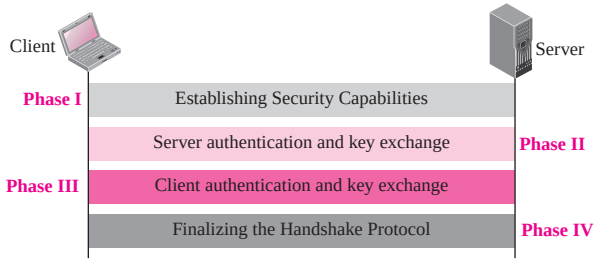

The Record Protocol is the carrier. It carries messages from three other protocols as well as the data coming from the application layer. Messages from the Record Protocol are payloads to the transport layer, normally TCP. The Handshake Protocol provides security parameters for the Record Protocol. It establishes a cipher set and provides keys and security parameters. It also authenticates the server to the client and the client

to the server if needed. The ChangeCipherSpec Protocol is used for signaling the readiness of cryptographic secrets. The Alert Protocol is used to report abnormal conditions. We will briefly discuss these protocols in this section.

### *Handshake Protocol*

The **Handshake Protocol** uses messages to negotiate the cipher suite, to authenticate the server to the client and the client to the server if needed, and to exchange information for building the cryptographic secrets. The handshaking is done in four phases, as shown in Figure 30.20.

**Figure 30.20** *Handshake Protocol*




**Phase I: Establishing Security Capability** In Phase I, the client and the server announce their security capabilities and choose those that are convenient for both. In this phase, a session ID is established and the cipher suite is chosen. The parties agree upon a particular compression method. Finally, two random numbers are selected, one by the client and one by the server, to be used for creating a master secret as we saw before.

**After Phase I, the client and server know the version of SSL, the cryptographic algorithms, the compression method, and the two random numbers for key generation.**

**Phase II: Server Key Exchange and Authentication** In Phase II, the server authenticates itself if needed. The sender may send its certificate, its public key, and may also request certificates from the client.

**After Phase II, the server is authenticated to the client, and the client knows the public key of the server if required.**

**Phase III: Client Key Exchange and Authentication** Phase III is designed to authenticate the client.

**After Phase III, The client is authenticated for the serve, and both the client and the server know the pre-master secret.**

**Phase IV: Finalizing and Finishing** In Phase IV, the client and server send messages to change cipher specification and to finish the handshaking protocol.

### *ChangeCipherSpec Protocol*

We have seen that the negotiation of the cipher suite and the generation of cryptographic secrets are formed gradually during the Handshake Protocol. The question now is: When can the two parties use these parameter secrets? SSL mandates that the parties cannot use these parameters or secrets until they have sent or received a special message, the ChangeCipherSpec message, which is exchanged during the Handshake protocol and defined in the **ChangeCipherSpec Protocol.** The reason is that the issue is not just sending or receiving a message. The sender and the receiver need two states, not one. One state, the pending state, keeps track of the parameters and secrets. The other state, the active state, holds parameters and secrets used by the Record Protocol to sign/verify or encrypt/decrypt messages. In addition, each state holds two sets of values: *read* (inbound) and *write* (outbound).

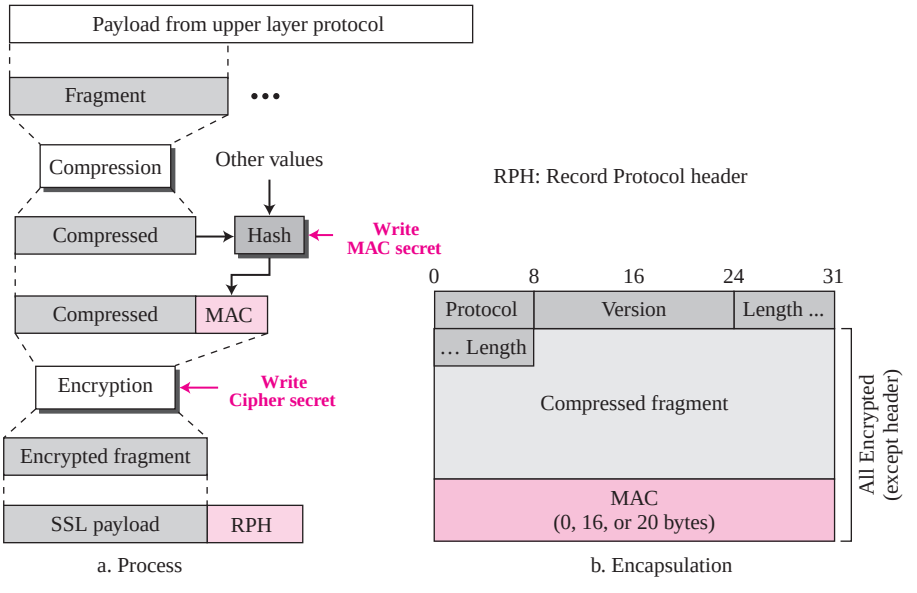
### *Alert Protocol*

SSL uses the **Alert Protocol** for reporting errors and abnormal conditions. It uses only one message that describes the problem and its level (warning or fatal).

### *Record Protocol*

The **Record Protocol** carries messages from the upper layer (Handshake Protocol, ChangeCipherSpec Protocol, Alert Protocol, or application layer). The message is fragmented and optionally compressed; a MAC is added to the compressed message using the negotiated hash algorithm. The compressed fragment and the MAC are encrypted using the negotiated encryption algorithm. Finally, the SSL header is added to the encrypted message. Figure 30.21 shows this process at the sender. The process at the receiver is reversed.

**Figure 30.21** *Processing done by the Record Protocol*



## 30.3 APPLICATION LAYER SECURITY

This section discusses two protocols providing security services for e-mails: Pretty Good Privacy (PGP) and Secure/Multipurpose Internet Mail Extension (S/MIME).

### E-mail Security

Sending an e-mail is a one-time activity. The nature of this activity is different from those we saw in the previous two sections. In IPSec or SSL, we assume that the two parties create a session between themselves and exchange data in both directions. In e-mail, there is no session. Alice and Bob cannot create a session. Alice sends a message to Bob; sometime later, Bob reads the message and may or may not send a reply. We discuss the security of a unidirectional message because what Alice sends to Bob is totally independent from what Bob sends to Alice.

#### *Cryptographic Algorithms*

If e-mail is a one-time activity, how can the sender and receiver agree on a cryptographic algorithm to use for e-mail security? If there is no session and no handshaking to negotiate the algorithms for encryption/decryption and hashing, how can the receiver know which algorithm the sender has chosen for each purpose?

To solve the problem, the protocol defines a set of algorithms for each operation that the user used in his/her system. Alice includes the name (or identifiers) of the algorithms she has used in the e-mail. For example, Alice can choose DES for encryption/decryption and MD5 for hashing. When Alice sends a message to Bob, she includes the corresponding identifiers for DES and MD5 in her message. Bob receives the message and extracts the identifiers first. He then knows which algorithm to use for decryption and which one for hashing.

**In e-mail security, the sender of the message needs to include the name or identifiers of the algorithms used in the message.**

#### *Cryptographic Secrets*

The same problem for the cryptographic algorithms applies to the cryptographic secrets (keys). If there is no negotiation, how can the two parties establish secrets between themselves? The e-mail security protocols today require that encryption/decryption be done using a symmetric-key algorithm and a one-time secret key sent with the message. Alice can create a secret key and send it with the message she sends to Bob. To protect the secret key from interception by Eve, the secret key is encrypted with Bob's public key. In other words, the secret key itself is encrypted.

**In e-mail security, the encryption/decryption is done using a symmetric-key algorithm, but the secret key to decrypt the message is encrypted with the public key of the receiver and is sent with the message.**

### *Certificates*

One more issue needs to be considered before we discuss any e-mail security protocol in particular. It is obvious that some public-key algorithms must be used for e-mail security. For example, we need to encrypt the secret key or sign the message. To encrypt the secret key, Alice needs Bob's public key; to verify a signed message, Bob needs Alice's public key. So, for sending a small authenticated and confidential message, two public keys are needed. How can Alice be assured of Bob's public key, and how can Bob be assured of Alice's public key? Each e-mail security protocol has a different method of certifying keys.

## Pretty Good Privacy (PGP)

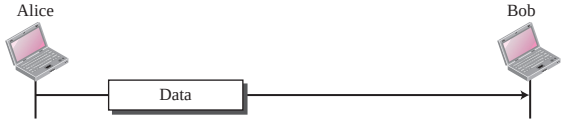
The first protocol discussed in this chapter is called **Pretty Good Privacy (PGP).** PGP was invented by Phil Zimmermann to provide e-mail with privacy, integrity, and authentication. PGP can be used to create a secure e-mail messages.

### *Scenarios*

Let us first discuss the general idea of PGP, moving from a simple scenario to a complex one. We use the term "Data" to show the message prior to processing.

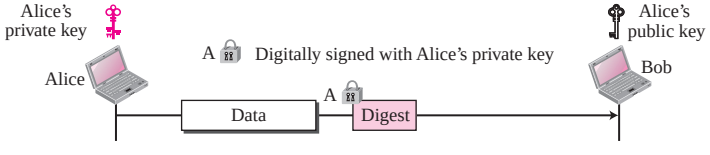
**Plaintext** The simplest scenario is to send the e-mail message in plaintext as shown in Figure 30.22. There is no message integrity or confidentiality in this scenario.

**Figure 30.22** *A plaintext message*




**Message Integrity** Probably the next improvement is to let Alice sign the message. Alice creates a digest of the message and signs it with her private key. Figure 30.23 shows the situation.

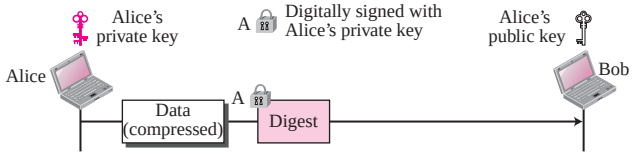
**Figure 30.23** *An authenticated message*




When Bob receives the message, he verifies the message by using Alice's public key. Two keys are needed for this scenario. Alice needs to know her private key; Bob needs to know Alice's public key.

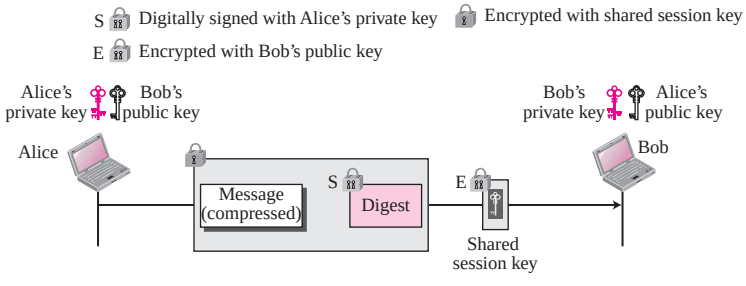
**Compression** A further improvement is to compress the message to make the packet more compact. This improvement has no security benefit, but it eases the traffic. Figure 30.24 shows the new scenario.

**Figure 30.24** *A compressed message*




**Confidentiality with One-Time Session Key** Figure 30.25 shows the situation. As we discussed before, confidentiality in an e-mail system can be achieved using conventional encryption with a one-time session key. Alice can create a session key, use the session key to encrypt the message and the digest, and send the key itself with the message. However, to protect the session key, Alice encrypts it with Bob's public key.

**Figure 30.25** *A confidential message*




When Bob receives the packet, he first decrypts the key, using his private key to remove the key. He then uses the session key to decrypt the rest of the message. After decompressing the rest of the message, Bob creates a digest of the message and checks to see if it is equal to the digest sent by Alice. If it is, then the message is authentic.

**Code Conversion** Another service provided by PGP is code conversion. Most e-mail systems allow the message to consist of only ASCII characters. To translate other characters not in the ASCII set, PGP uses Radix-64 conversion (see Chapter 23).

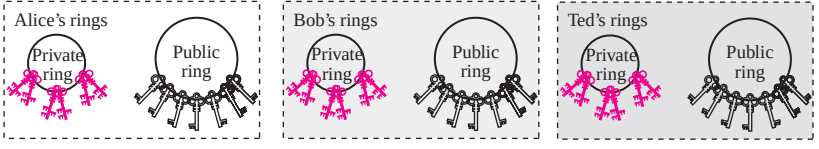
### *Segmentation*

PGP allows segmentation of the message after it has been converted to Radix-64 to make each transmitted unit the uniform size allowed by the underlying e-mail protocol.

## Key Rings

In all previous scenarios, we assumed that Alice needs to send a message only to Bob. That is not always the case. Alice may need to send messages to many people; she needs **key rings.** In this case, Alice needs a ring of public keys, with a key belonging to each person with whom Alice needs to correspond (send or receive messages). In addition, the PGP designers specified a ring of private/public keys. One reason is that Alice may wish to change her pair of keys from time to time. Another reason is that Alice may need to correspond with different groups of people (friends, colleagues, and so on). Alice may wish to use a different key pair for each group. Therefore, each user needs to have two sets of rings: a ring of private/public keys and a ring of public keys of other people. Figure 30.26 shows a community of four people, each having a ring of pairs of private/public keys and, at the same time, a ring of public keys belonging to other people in the community.

**Figure 30.26** *Key rings in PGP*




Alice, for example, has several pairs of private/public keys belonging to her and public keys belonging to other people. Note that everyone can have more than one public key. Two cases may arise.

1. Alice needs to send a message to another person in the community.
   - **a.** She uses her private key to sign the digest.
   - **b.** She uses the receiver's public key to encrypt a newly created session key.
   - **c.** She encrypts the message and signed digest with the session key created.
2. Alice receives a message from another person in the community.
   - **a.** She uses her private key to decrypt the session key.
   - **b.** She uses the session key to decrypt the message and digest.
   - **c.** She uses her public key to verify the digest.

### *PGP Algorithms*

PGP defines a set of asymmetric-key and symmetric-key algorithms, cryptography hash functions, and compression methods. We leave the details of these algorithm to the books devoted to PGP. When Alice sends an e-mail to Bob, she defines the algorithm she has used for each purpose.

## PGP Certificates

PGP, like other protocols we have seen so far, uses certificates to authenticate public keys. However, the process is totally different, as explained below.

### PGP Certificates

In PGP, there is no need for CAs; anyone in the ring can sign a certificate for anyone else in the ring. Bob can sign a certificate for Ted, John, Anne, and so on. There is no hierarchy of trust in PGP; there is no tree. The lack of hierarchical structure may result in the fact that Ted may have one certificate from Bob and another certificate from Liz. If Alice wants to follow the line of certificates for Ted, there are two paths: one starts from Bob and one starts from Liz. An interesting point is that Alice may fully trust Bob, but only partially trust Liz. There can be multiple paths in the line of trust from a fully or partially trusted authority to a certificate. In PGP, the issuer of a certificate is usually called an *introducer*.

**In PGP, there can be multiple paths from fully or partially trusted authorities to any subject.**

### Trusts and Legitimacy

The entire operation of PGP is based on introducer trust, the certificate trust, and the legitimacy of the public keys.

**Introducer Trust Levels** With the lack of a central authority, it is obvious that the ring cannot be very large if every user in the PGP ring of users has to fully trust everyone else. (Even in real life we cannot fully trust everyone that we know.) To solve this problem, PGP allows different levels of trust. The number of levels is mostly implementation dependent, but for simplicity, let us assign three levels of trust to any introducer: *none*, *partial*, and *full*. The introducer trust level specifies the trust levels issued by the introducer for other people in the ring. For example, Alice may fully trust Bob, partially trust Anne, and not trust John at all. There is no mechanism in PGP to determine how to make a decision about the trustworthiness of the introducer; it is up to the user to make this decision.

**Certificate Trust Levels** When Alice receives a certificate from an introducer, she stores the certificate under the name of the subject (certified entity). She assigns a level of trust to this certificate. The certificate trust level is normally the same as the introducer trust level that issued the certificate. Assume that Alice fully trusts Bob, partially trusts Anne and Janette, and has no trust in John. The following scenarios can happen.

1. Bob issues two certificates, one for Linda (with public key K1) and one for Lesley (with public key K2). Alice stores the public key and certificate for Linda under Linda's name and assigns a *full* level of trust to this certificate. Alice also stores the certificate and public key for Lesley under Lesley's name and assigns a full level of trust to this certificate.
2. Anne issues a certificate for John (with public key K3). Alice stores this certificate and public key under John's name, but assigns a *partial* level for this certificate.
3. Janette issues two certificates, one for John (with public key K3) and one for Lee (with public key K4). Alice stores John's certificate under his name and Lee's certificate under his name, each with a *partial* level of trust. Note that John now has two certificates, one from Anne and one from Janette, each with a *partial* level of trust.
4. John issues a certificate for Liz. Alice can discard or keep this certificate with a signature trust of *none*.

**Key Legitimacy** The purpose of using introducer and certificate trusts is to determine the legitimacy of a public key. Alice needs to know how legitimate the public keys of Bob, John, Liz, Anne, and so on are. PGP defines a very clear procedure for determining key legitimacy. The level of the key legitimacy for a user is the weighted trust levels of that user. For example, suppose we assign the following weights to certificate trust levels:

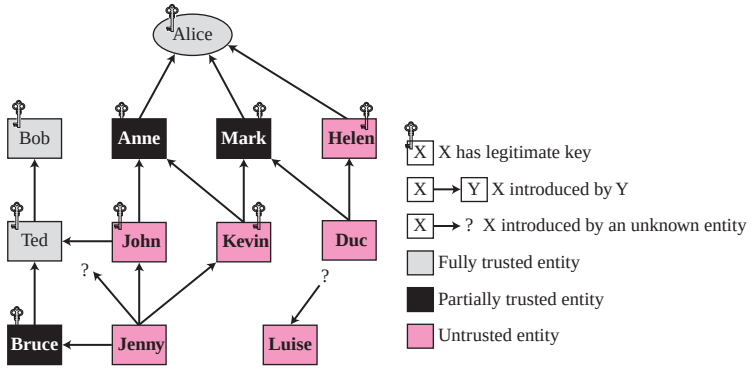
1. A weight of 0 to a nontrusted certificate
2. A weight of 1/2 to a certificate with partial trust
3. A weight of 1 to a certificate with full trust

Then to fully trust an entity, Alice needs one fully trusted certificate or two partially trusted certificates for that entity. For example, Alice can use John's public key in the previous scenario because both Anne and Janette have issued a certificate for John, each with a certificate trust level of 1/2. Note that the legitimacy of a public key belonging to an entity does not have anything to do with the trust level of that person. Although Bob can use John's public key to send a message to him, Alice cannot accept any certificate issued by John because, for Alice, John has a trust level of *none*.

### *Trust Model in PGP*

As Zimmermann has proposed, we can create a trust model for any user in a ring with the user as the center of activity. Such a model can look like the one shown in Figure 30.27. The figure shows the trust model for Alice at some moment.

**Figure 30.27** *Trust model*




Let us elaborate on the figure. Figure 30.27 shows that there are three entities in Alice's ring with full trust (Alice herself, Bob, and Ted). The figure also shows three entities with partial trust (Anne, Mark, and Bruce). There are also six entities with no trust. Nine entities have a legitimate key. Alice can encrypt a message to any one of these entities or verify a signature received from one of these entities (Alice's key is never used in this model). There are also three entities that do not have any legitimate keys with Alice.

Bob, Anne, and Mark have made their keys legitimate by sending their keys by e-mail and verifying their fingerprints by phone. Helen, on the other hand, has sent a certificate from a CA because she is not trusted by Alice and verification on the phone is not possible. Although Ted is fully trusted, he has given Alice a certificate signed by Bob. John has sent Alice two certificates, one signed by Ted and one by Anne. Kevin has sent two certificates to Alice, one signed by Anne and one by Mark. Each of these certificates gives Kevin half a point of legitimacy; therefore, Kevin's key is legitimate. Duc has sent two certificates to Alice, one signed by Mark and the other by Helen. Since Mark is half-trusted and Helen is not trusted, Duc does not have a legitimate key. Jenny has sent four certificates, one signed by a half-trusted entity, two by untrusted entities, and one by an unknown entity. Jenny does not have enough points to make her key legitimate. Luise has sent one certificate signed by an unknown entity. Note that Alice may keep Luise's name in the table in case future certificates for Luise arrive.

### *Web of Trust*

PGP can eventually make a **web of trust** between a group of people. If each entity introduces more entities to other entities, the public key ring for each entity gets larger and larger and entities in the ring can send secure e-mail to each other.

### *Key Revocation*

It may become necessary for an entity to revoke his or her public key from the ring. This may happen if the owner of the key feels that the key is compromised (stolen, for example) or just too old to be safe. To revoke a key, the owner can send a revocation certificate signed by herself. The revocation certificate must be signed by the old key and disseminated to all the people in the ring that use that public key.

### *PGP Packets*

A message in PGP consists of one or more packets. During the evolution of PGP, the format and the number of packet types have changed. We do not discuss the formats of these packets here.

### *Applications of PGP*

PGP has been extensively used for personal e-mails. It will probably continue to be.

## S/MIME

Another security service designed for electronic mail is **Secure/Multipurpose Internet Mail Extension (S/MIME).** The protocol is an enhancement of the Multipurpose Internet Mail Extension (MIME) protocol we discussed in Chapter 23.

### *Cryptographic Message Syntax (CMS)*

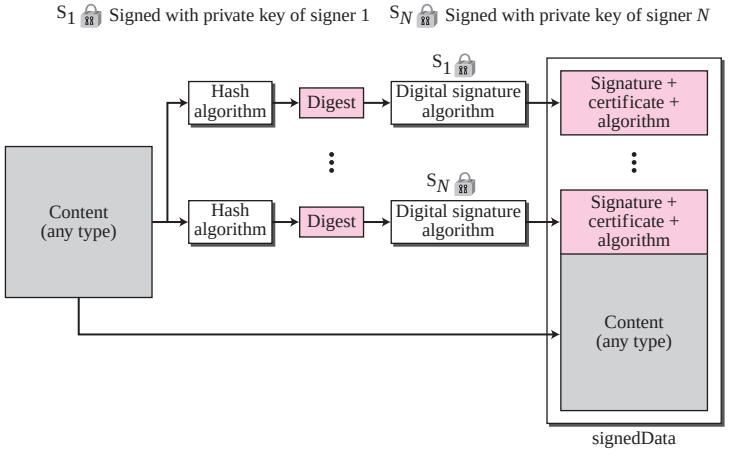
To define how security services, such as confidentiality or integrity, can be added to MIME content types, S/MIME has defined **Cryptographic Message Syntax (CMS).** The syntax in each case defines the exact encoding scheme for each content type. The following describe the type of message and different subtypes that are created from these messages. For details, the reader is referred to RFC 3369 and RFC 3370.

**Data Content Type** This is an arbitrary string. The object created is called *Data*.

**Signed-Data Content Type** This type provides only integrity of data. It contains any type and zero or more signature values. The encoded result is an *object* called *signedData*. Figure 30.28 shows the process of creating an object of this type. The following are the steps in the process:

1. For each signer, a message digest is created from the content using the specific hash algorithm chosen by that signer.
2. Each message digest is signed with the private key of the signer.
3. The content, signature values, certificates, and algorithms are then collected to create the *signedData* object.

**Figure 30.28** *Signed-data content type*




**Enveloped-Data Content Type** This type is used to provide privacy for the message. It contains any type and zero or more encrypted keys and certificates. The encoded result is an *object* called *envelopedData*. Figure 30.29 shows the process of creating an object of this type.

1. A pseudorandom session key is created for the symmetric-key algorithms to be used.
2. For each recipient, a copy of the session key is encrypted with the public key of each recipient.
3. The content is encrypted using the defined algorithm and created session key.
4. The encrypted contents, encrypted session keys, algorithm used, and certificates are encoded using Radix-64.

**Digested-Data Content Type** This type is used to provide integrity for the message. The result is normally used as the content for the enveloped-data content type. The encoded result is an *object* called *digestedData*. Figure 30.30 shows the process of creating an object of this type.

**Figure 30.29** *Enveloped-data content type*

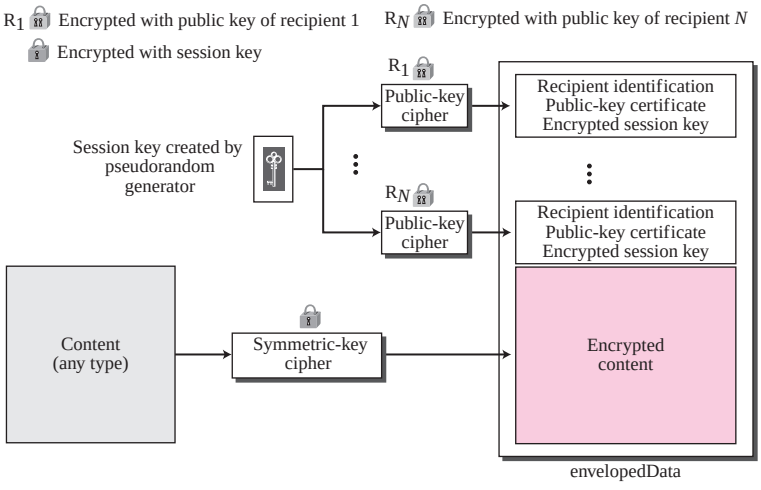
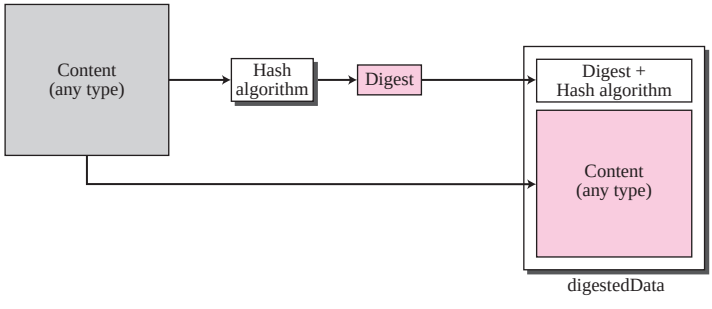



**Figure 30.30** *Digest-data content type*




1. A message digest is calculated from the content.
2. The message digest, the algorithm, and the content are added together to create the *digestedData* object.

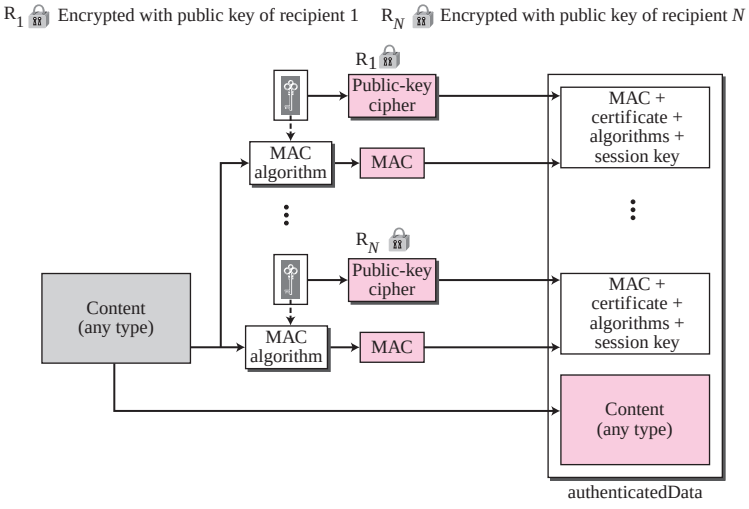
**Encrypted-Data Content Type** This type is used to create an encrypted version of any content type. Although this looks like the enveloped-data content type, the encrypted-data content type has no recipient. It can be used to store the encrypted data instead of transmitting it. The process is very simple; the user employs any key (normally driven from the password) and any algorithm to encrypt the content. The encrypted content is stored without including the key or the algorithm. The object created is called *encryptedData*.

**Authenticated-Data Content Type** This type is used to provide authentication of the data. The object is called *authenticatedData*. Figure 30.31 shows the process.

1. Using a pseudorandom generator, a MAC key is generated for each recipient.

2. The MAC key is encrypted with the public key of the recipient.
3. A MAC is created for the content.
4. The content, MAC, algorithms, and other informations are collected together to form the authenticatedData object.

**Figure 30.31** *Authenticated-data content type*




### *Key Management*

The key management in S/MIME is a combination of key management used by X.509 and PGP. S/MIME uses public-key certificates signed by the certificate authorities defined by X.509. However, the user is responsible to maintain the web of trust to verify signatures as defined by PGP.

### *Cryptographic Algorithms*

S/MIME defines several cryptographic algorithms. We leave the details of these algorithms to the books dedicated to security in the Internet.

### Example 30.1

The following shows an example of an enveloped-data in which a small message is encrypted using triple DES.

**Content-Type: application/pkcs7-mime; mime-type=enveloped-data**
**Content-Transfer-Encoding: Radix-64**
**Content-Description: attachment**
**name="report.txt";**
cb32ut67f4bhijHU21oi87eryb0287hmnklsgFDoY8bc659GhIGfH6543mhjkdsaH23YjBnmN
ybmlkzjhgfdyhGe23Kjk34XiuD678Es16se09jy76jHuytTMDcbnmlkjgfFdiuyu678543m0n3h
G34un12P2454Hoi87e2ryb0H2MjN6KuyrlsgFDoY897fk923jljk1301XiuD6gh78EsUyT23y

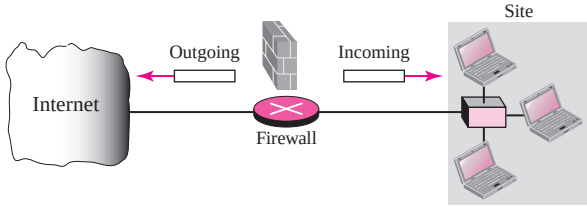
### Applications of S/MIME

It is predicted that S/MIME will become the industry choice to provide security for commercial e-mail.

## 30.4 FIREWALLS

All previous security measures cannot prevent Eve from sending a harmful message to a system. To control access to a system we need firewalls. A **firewall** is a device (usually a router or a computer) installed between the internal network of an organization and the rest of the Internet. It is designed to forward some packets and filter (not forward) others. Figure 30.32 shows a firewall.

**Figure 30.32** *Firewall*




For example, a firewall may filter all incoming packets destined for a specific host or a specific server such as HTTP. A firewall can be used to deny access to a specific host or a specific service in the organization. A firewall is usually classified as a packet-filter firewall or a proxy-based firewall.

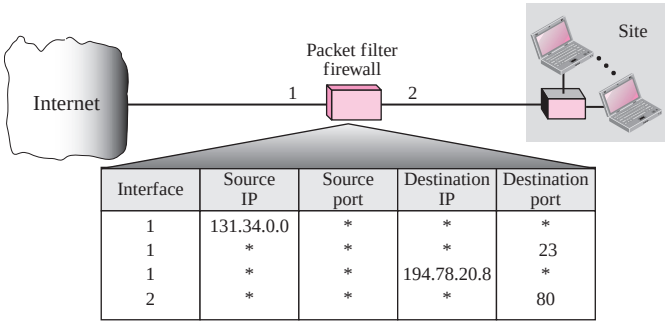
### Packet-Filter Firewall

A firewall can be used as a packet filter. It can forward or block packets based on the information in the network layer and transport layer headers: source and destination IP addresses, source and destination port addresses, and type of protocol (TCP or UDP). A **packet-filter firewall** is a router that uses a filtering table to decide which packets must be discarded (not forwarded). Figure 30.33 shows an example of a filtering table for this kind of a firewall.

According to the figure, the following packets are filtered:

1. Incoming packets from network 131.34.0.0. are blocked (security precaution). Note that the * (asterisk) means "any."
2. Incoming packets destined for any internal TELNET server (port 23) are blocked.

**Figure 30.33** *Packet-filter firewall*




| Interface | Source IP | Source port | Destination IP | Destination port |
|---|---|---|---|---|
| 1 | 131.34.0.0 | * | * | * |
| 1 | * | * | * | 23 |
| 1 | * | * | 194.78.20.8 | * |
| 2 | * | * | * | 80 |

3. Incoming packets destined for internal host 194.78.20.8. are blocked. The organization wants this host for internal use only.
4. Outgoing packets destined for an HTTP server (port 80) are blocked. The organization does not want employees to browse the Internet.

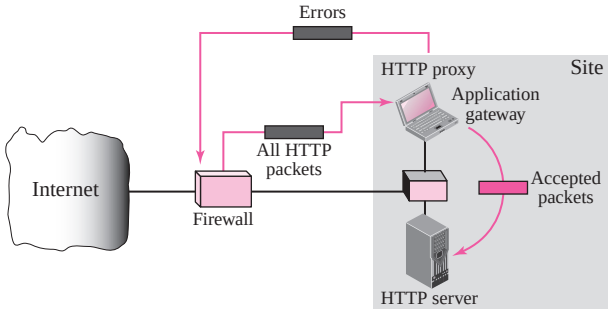
**A packet-filter firewall filters at the network or transport layer.**

## Proxy Firewall

The packet-filter firewall is based on the information available in the network layer and transport layer headers (IP and TCP/UDP). However, sometimes we need to filter a message based on the information available in the message itself (at the application layer). As an example, assume that an organization wants to implement the following policies regarding its Web pages: only those Internet users who have previously established business relations with the company can have access; access to other users must be blocked. In this case, a packet-filter firewall is not feasible because it cannot distinguish between different packets arriving at TCP port 80 (HTTP). Testing must be done at the application level (using URLs).

One solution is to install a proxy computer (sometimes called an **application gateway**), which stands between the customer computer and the corporation computer. When the user client process sends a message, the application gateway runs a server process to receive the request. The server opens the packet at the application level and finds out if the request is legitimate. If it is, the server acts as a client process and sends the message to the real server in the corporation. If it is not, the message is dropped and an error message is sent to the external user. In this way, the requests of the external users are filtered based on the contents at the application layer. Figure 30.34 shows an application gateway implementation for HTTP.

**Figure 30.34** *Proxy firewall*




**A proxy firewall filters at the application layer.**

## 30.5 RECOMMENDED READING

The following books give more details about subjects discussed in this chapter. The items in brackets refer to the reference list at the end of the text. In particular, we recommend [For 08], [Sta 06], [Bis 05], [Mao 04], [Sti 06], [Res 01], [Tho 00], [Dor & Har 03], and [Gar 01].

## 30.6 KEY TERMS

Alert Protocol
application gateway
Authentication Header (AH) Protocol
ChangeCipherSpec Protocol
cipher suite
connection
Cryptographic Message Syntax (CMS)
Encapsulating Security Payload (ESP)
Extension (S/MIME)
firewall
Handshake Protocol
Internet Key Exchange (IKE)
Internet Security Association and Key Management Protocol (ISAKMP)
IP Security (IPSec)
key material
key ring
master secret
Oakley
packet-filter firewall
pre-master secret
Pretty Good Privacy (PGP)
Record Protocol
Secure Sockets Layer (SSL) protocol
Secure/Multipurpose Internet Mail Extension (S/MIME)
Security Association (SA)
Security Association Database (SAD)
Security Policy (SP)
Security Policy Database (SPD)
session
SKEME
Transport Layer Security (TLS) protocol
transport mode
tunnel mode
virtual private network (VPN)
web of trust

## 30.7 SUMMARY

- IP Security (IPSec) is a collection of protocols designed by the IETF to provide security for a packet at the network level. IPSec operates in transport or tunnel mode. IPSec defines two protocols: Authentication Header (AH) Protocol and Encapsulating Security Payload (ESP) Protocol to provide authentication and encryption or both for packets at the IP level. A virtual private network is an implementation of IPSec that provides privacy for organization with multiple sites.
- A transport layer security protocol provides end-to-end security services for applications that use the services of a reliable transport layer protocol such as TCP. Two protocols are dominant today for providing security at the transport layer: Secure Sockets Layer (SSL) and Transport Layer Security (TLS). SSL (or TLS) provides services such as fragmentation, compression, message integrity, confidentiality, and framing on data received from the application layer.
- The Pretty Good Privacy (PGP), invented by Phil Zimmermann, provides e-mail with privacy, integrity, and authentication. Another security service designed for electronic mail is Secure/Multipurpose Internet Mail Extension (S/MIME). The protocol is an enhancement of the Multipurpose Internet Mail Extension (MIME) protocol.
- A firewall is a device (usually a router or a computer) installed between the internal network of an organization and the rest of the Internet. It is designed to forward some packets and filter others. A firewall is usually classified as a packet-filter firewall or a proxy-based firewall. The packet-filter firewall is based on the information available in the network layer and transport layer headers. The proxy firewall is based on the information available at the application layer.

## 30.8 PRACTICE SET

### Exercises

1. Host A and host B use IPSec in the transport mode. Can we say that the two hosts need to create a virtual connection-oriented service between them? Explain.
2. When we talk about authentication in IPSec, do we mean *message authentication* or *entity authentication*? Explain.
3. When we talk about authentication in SSL, do we mean *message authentication* or *entity authentication*? Explain.
4. When we talk about authentication in PGP (or S/MIME), do we mean *message authentication* or *entity authentication*? Explain.
5. If cryptography algorithms in PGP or S/MIME cannot be negotiated, how can the receiver of the e-mail determine which algorithm have been used by the sender?
6. Can we use SSL with UDP? Explain.
7. Why is there no need for security association with SSL?

8. Compare and contrast PGP and S/MIME. What are the advantage and disadvantage of each?
9. If Alice and Bob are continuously sending messages to each other, can they create a security association once and use it for every packet exchanged? Explain.
10. Should the handshaking in SSL occur before or after the three-way handshaking in TCP? Can they be combined? Explain.

## Research Activities

11. We discuss security services only for SMTP at the application layer. Are there any security services for other application-layer protocols?
12. Use the literature and the Internet to find more about IKE.
13. Use the literature and the Internet to find more about handshake protocol in SSL.
14. Use the literature and the Internet to find more about TLS.

# PART 7

# Appendices

APPENDIX

# A

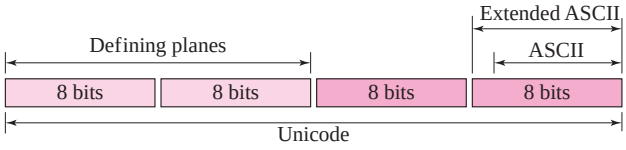
# *Unicode*

Computers use numbers. They store characters by assigning a number for each one. The original coding system was called ASCII (American Standard Code for Information Interchange) and had 128 numbers (0 to 127) each stored as a 7-bit number. ASCII could satisfactorily handle lowercase and uppercase letters, digits, punctuation characters, and some control characters. An attempt was made to extend the ASCII character set to 8 bits. The new code, which was called Extended ASCII, was never internationally standardized.

To overcome the difficulties inherent in ASCII and Extended ASCII, the Unicode Consortium (a group of multilingual software manufacturers) created a universal encoding system to provide a comprehensive character set called **Unicode.**

Unicode was originally a 2-byte character set. Unicode version 3, however, is a 4-byte code and is fully compatible with ASCII and Extended ASCII. The ASCII set, which is now called *Basic Latin*, is Unicode with the upper 25 bits set to zero. Extended ASCII, which is now called Latin-1, is Unicode with the 24 upper bits set to zero. Figure A.1 shows how the different systems are compatible.

**Figure A.1** *Unicode compatibility*




Each character or symbol in this code is defined by a 32-bit number. The code can define up to $2^{32}$ (4,294,967,296) characters or symbols. The notation uses hexadecimal digits in the following format:

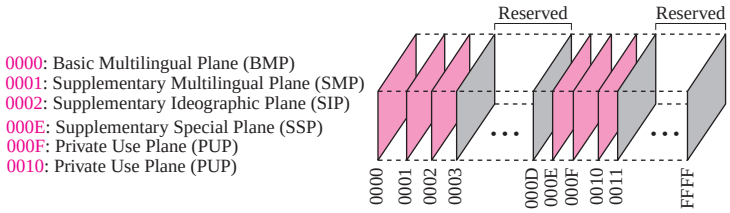
**U-XXXXXXXX**

Each X is a hexadecimal digit. Therefore, the numbering goes from U-00000000 to U-FFFFFFFF.

## A.1 PLANES

Unicode divides the available space codes into planes. The most significant 16 bits define the plane, which means we can have 65,536 planes. Each plane can define up to 65,536 characters or symbols. Figure A.2 shows the structure of Unicode spaces and planes.

**Figure A.2** *Unicode planes*




### Basic Multilingual Plane (BMP)

Plane $(0000)_{16}$, the basic multilingual plane (BMP), is designed to be compatible with the previous 16-bit Unicode. The most significant 16 bits in this plane are all zeros. The codes are normally shown as U+XXXX with the understanding that XXXX defines only the least significant 16 bits. This plane mostly defines character sets in different languages with the exception of some codes used for control or other special characters.

### Other Planes

There are some other (non-reserved) planes, that we briefly describe below:

#### *Supplementary Multilingual Plane (SMP)*

Plane $(0001)_{16}$, the supplementary multilingual plane (SMP), is designed to provide more codes for those multilingual characters that are not included in the BMP.

#### *Supplementary Ideographic Plane (SIP)*

Plane $(0002)_{16}$, the supplementary ideographic plane (SIP), is designed to provide codes for ideographic symbols, symbols that primarily denote an idea (or meaning) in contrast to a sound (or pronunciation).

#### *Supplementary Special Plane (SSP)*

Plane $(000E)_{16}$, the supplementary special plane (SSP), is used for special characters.

#### *Private Use Planes (PUPs)*

Planes (000F) and $(0010)_{16}$, private use planes (PUPs), are for private use.

## A.2 ASCII

The American Standard Code for Information Interchange (ASCII) is a 7-bit code that was designed to provide code for 128 symbols, mostly in American English. Today, ASCII, or Basic Latin, is part of Unicode. It occupies the first 128 codes in Unicode (00000000 to 0000007F). Table A.1 contains the hexadecimal and graphic codes (symbols). The codes in hexadecimal just define the two least significant digits in Unicode. To find the actual code, we prepend 000000 in hexadecimal to the code.

**Table A.1** *ASCII Codes*

| *Hex* | *Symbol* | *Hex* | *Symbol* | *Hex* | *Symbol* | *Hex* | *Symbol* |
|---|---|---|---|---|---|---|---|
| 00 | NULL | 20 | SP | 40 | @ | 60 | ` |
| 01 | SOH | 21 | ! | 41 | A | 61 | a |
| 02 | STX | 22 | " | 42 | B | 62 | b |
| 03 | ETX | 23 | # | 43 | C | 63 | c |
| 04 | EOT | 24 | $ | 44 | D | 64 | d |
| 05 | ENQ | 25 | % | 45 | E | 65 | e |
| 06 | ACK | 26 | & | 46 | F | 66 | f |
| 07 | BEL | 27 | ' | 47 | G | 67 | g |
| 08 | BS | 28 | ( | 48 | H | 68 | h |
| 09 | HT | 29 | ) | 49 | I | 69 | i |
| 0A | LF | 2A | * | 4A | J | 6A | j |
| 0B | VT | 2B | + | 4B | K | 6B | k |
| 0C | FF | 2C | , | 4C | L | 6C | l |
| 0D | CR | 2D | – | 4D | M | 6D | m |
| 0E | SO | 2E | . | 4E | N | 6E | n |
| 0F | SI | 2F | / | 4F | O | 6F | o |
| 10 | DLE | 30 | 0 | 50 | P | 70 | p |
| 11 | DC1 | 31 | 1 | 51 | Q | 71 | q |
| 12 | DC2 | 32 | 2 | 52 | R | 72 | r |
| 13 | DC3 | 33 | 3 | 53 | S | 73 | s |
| 14 | DC4 | 34 | 4 | 54 | T | 74 | t |
| 15 | NAK | 35 | 5 | 55 | U | 75 | u |
| 16 | SYN | 36 | 6 | 56 | V | 76 | v |
| 17 | ETB | 37 | 7 | 57 | W | 77 | w |
| 18 | CAN | 38 | 8 | 58 | X | 78 | x |
| 19 | EM | 39 | 9 | 59 | Y | 79 | y |
| 1A | SUB | 3A | : | 5A | Z | 7A | z |
| 1B | ESC | 3B | ; | 5B | [ | 7B | { |
| 1C | FS | 3C | < | 5C | \ | 7C | \| |
| 1D | GS | 3D | = | 5D | ] | 7D | } |
| 1E | RS | 3E | > | 5E | ^ | 7E | ~ |
| 1F | US | 3F | ? | 5F | _ | 7F | DEL |

## Some Properties of ASCII

ASCII has some interesting properties that we briefly mention here.

1. The space character $(20)_{16}$, is a printable character. It prints a blank space.
2. The uppercase letters start from $(41)_{16}$. The lowercase letters start from $(61)_{16}$. When compared, uppercase letters are numerically smaller than lowercase letters. This means that in a sorted list based on ASCII values, the uppercase letters appear before the lowercase letters.
3. The uppercase and lowercase letters differ by only one bit in the 7-bit code. For example, character *A* is $(1000001)_2$ and character *a* is $(1100001)_2$. The difference is in bit 6, which is 0 in uppercase letters and 1 in lowercase letters. If we know the code for one case, we can easily find the code for the other by adding or subtracting $(20)_{16}$, or we can just flip the sixth bit.
4. The uppercase letters are not immediately followed by lowercase letters. There are some punctuation characters in between.
5. Digits (0 to 9) start from $(30)_{16}$. This means that if you want to change a numeric character to its face value as an integer, you need to subtract $(30)_{16} = 48$ from it.
6. The first 32 characters, $(00)_{16}$ to $(1F)_{16}$, and the last character, $(7F)_{16}$, are non-printable characters. Character $(00)_{16}$ simply is used as a delimiter to define the end of string character. Character $(7F)_{16}$ is the delete character used by some programming languages to delete the previous character. The rest of the non-printable characters are referred to as control characters and used in data communication. Table A.2 gives the description of these characters.

**Table A.2** *ASCII Codes*

| *Symbol* | *Interpretation* | *Symbol* | *Interpretation* |
|---|---|---|---|
| SOH | Start of heading | DC1 | Device control 1 |
| STX | Start of text | DC2 | Device control 2 |
| ETX | End of text | DC3 | Device control 3 |
| EOT | End of transmission | DC4 | Device control 4 |
| ENQ | Enquiry | NAK | Negative acknowledgment |
| ACK | Acknowledgment | SYN | Synchronous idle |
| BEL | Ring bell | ETB | End of transmission block |
| BS | Backspace | CAN | Cancel |
| HT | Horizontal tab | EM | End of medium |
| LF | Line feed | SUB | Substitute |
| VT | Vertical tab | ESC | Escape |
| FF | Form feed | FS | File separator |
| CR | Carriage return | GS | Group separator |
| SO | Shift out | RS | Record separator |
| SI | Shift in | US | Unit separator |
| DLE | Data link escape | | |

APPENDIX

B

# Positional Numbering Systems

A **positional number system** uses a set of symbols. The value that each symbol represents, however, depends on its **face value** and its **place value,** the value associated with the position it occupies in the number, In other words, we have

Symbol value = Face value × Place value

Number value = Sum of Symbol values

In this appendix, we discuss only integers, numbers with no fractional part; the discussion of reals, numbers with a fractional part is similar.

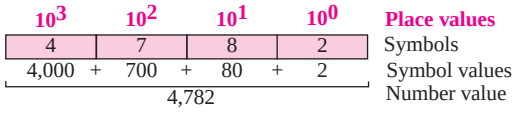
## B.1 DIFFERENT SYSTEMS

We first show how integers can be represented in four different systems: base 10, base 2, base 16, and base 256.

### Base 10: Decimal

The first positional system we discuss is called the **decimal system.** The term *decimal* is derived from the Latin root *decem* (meaning *ten*). The decimal system uses 10 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9) with the same face values as the symbols. The place values in the decimal number system are powers of 10. Figure B.1 shows the place values and the symbol values in the integer 4,782.

**Figure B.1** *An example of a decimal number*

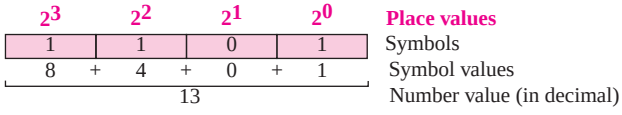



**The decimal system uses 10 symbols in which the place values are powers of 10.**

## Base 2: Binary

The second positional system we discuss is called the **binary system.** The term *binary* is derived from the Latin root *bi* (meaning *two by two*). The binary system uses 2 symbols (0 and 1) with the same face values as the symbols. The place values in the binary number system are powers of 2. Figure B.2 shows the place values and the symbol values in the binary $(1101)_2$. Note that we use subscript 2 to show that the number is in binary.

**Figure B.2** *An example of a binary number*

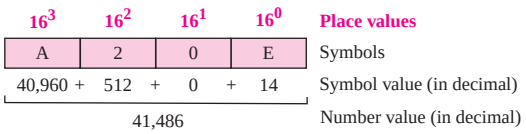



**The binary system uses 2 symbols in which the place values are powers of 2.**

### *Base 16: Hexadecimal*

The third positional system we discuss is called the **hexadecimal system.** The term *hexadecimal* is derived from the Greek root hex (meaning 6) and the Latin root *decem* (meaning *ten*). The hexadecimal system uses 16 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F). The face value of the ten symbols are the same as the symbols, but the face values of the symbols A to F are 10 to 15 respectively. The place values in the hexadecimal number system are powers of 16. Figure B.3 shows the place values and the symbol values in the hexadecimal $(A20E)_{16}$. Note that we use subscript 16 to show that the number is in hexadecimal.

**Figure B.3** *An example of a hexadecimal number*

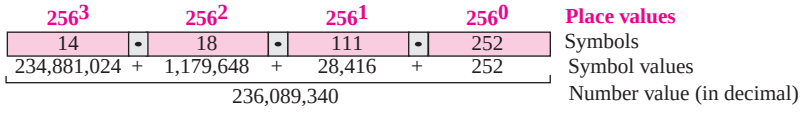



**The hexadecimal system uses 16 symbols in which the place values are powers of 16.**

### *Base 256: Dotted-Decimal Notation*

The fourth positional system we discuss is the base 256, which is called dotted-decimal notation. This system is used to represent IPv4 addressing. The place values in this system are powers of 256. However, since using 256 symbols is almost impossible, the symbols in this system are decimal numbers between 0 and 255, with the same face values as the symbols. To separate these numbers from each other, the system uses a dot as discussed in Chapter 5. Figure B.4 shows the place values and the symbol values of the address (14.18.111.252). Note that we never use more than four symbols in an IPv4 address.

**Figure B.4** *An example of a dotted-decimal notation*




**The dotted-decimal notations uses decimal numbers (0 to 255) as symbols, but inserts a dot between each symbol.**

## Comparison

Table B.1 shows how three different systems represent the decimal numbers 0 through 15. For example, decimal 13 is equivalent to binary 1101, which is equivalent to hexadecimal D.

**Table B.1** *Comparison of three systems*

| *Decimal* | *Binary* | *Hexadecimal* | *Decimal* | *Binary* | *Hexadecimal* |
|---|---|---|---|---|---|
| 0 | 0000 | 0 | 8 | 1000 | 8 |
| 1 | 0001 | 1 | 9 | 1001 | 9 |
| 2 | 0010 | 2 | 10 | 1010 | A |
| 3 | 0011 | 3 | 11 | 1011 | B |
| 4 | 0100 | 4 | 12 | 1100 | C |
| 5 | 0101 | 5 | 13 | 1101 | D |
| 6 | 0110 | 6 | 14 | 1110 | E |
| 7 | 0111 | 7 | 15 | 1111 | F |

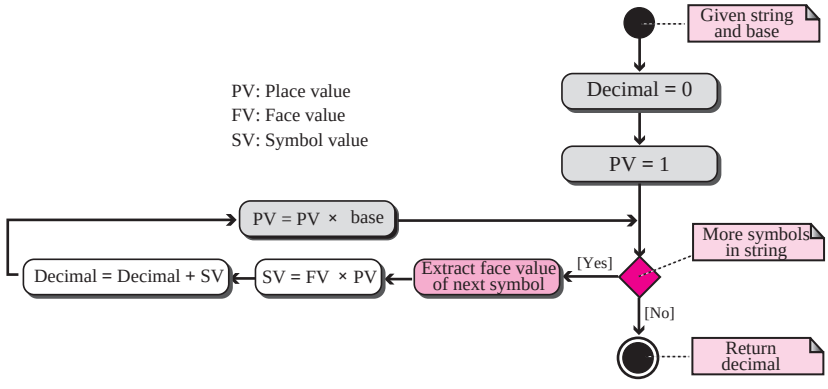
# B.2 CONVERSION

We need to know how to convert a number in one system to the equivalent number in another system.

## Conversion from Any Base to Decimal

Figures B.2 to B.4 actually show how we can manually convert a number in any base to decimal. However, it is easier to use the algorithm in Figure B.5. The algorithm uses the fact that the next place value is the previous value multiplied by the base (2, 16, or 256). The algorithm is a general one that can be used to convert a string of symbols in a given base to a decimal number. The only section in the algorithm that is different for each base is how to extract the next symbol in the string and find its face value. In the case of base 2, it is simple; the face value can be found by changing the symbol to a numeric value. In the case of base 16, we need to consider the case that the face value of symbol A is 10, the face value of symbol B is 11, and so on. In the case of base 256, we need to extract each string delimited by dots and change the string to its numerical value. We leave the details of these subalgorithms as an exercise because they are normally language-dependent.

**Figure B.5** *Algorithm to convert from any base to decimal.*




The manual implementation of the above algorithm for small numbers can be shown in a few examples.

### Example B.1

Show the equivalent of the binary number $(11100111)_2$ in decimal.

**Solution**

We follow the algorithm as shown below.:

| 128 | | 64 | | 32 | | 16 | | 8 | | 4 | | 2 | | 1 | Place values |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 1 | | 1 | | 0 | | 0 | | 1 | | 1 | | 1 | Face values |
| 128 | | 64 | | 32 | | 0 | | 0 | | 4 | | 2 | | 1 | Symbol values |
| **231** | | 103 | | 39 | | 7 | | 7 | | 7 | | 3 | | 1 | Decimal = 0 |

The value of decimal is initially set to 0. When the loop is terminated, the value of decimal is 231.

### Example B.2

Show the equivalent of the IPv4 address 12.14.67.24 in decimal.

**Solution**

We follow the algorithm as shown below.:

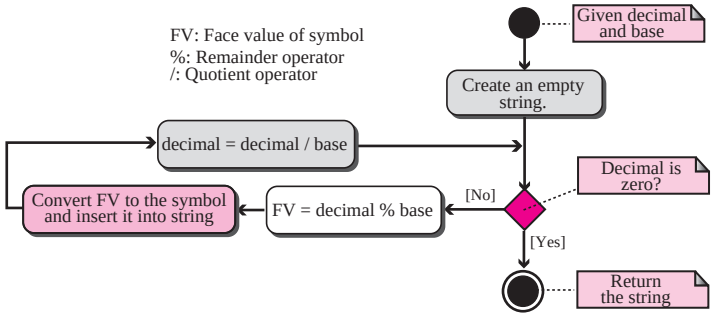
| 16,772,216 | | 65,536 | | 256 | | 1 | Place values |
|---|---|---|---|---|---|---|---|
| 12 | • | 14 | • | 67 | • | 24 | Face values |
| 201,266,592 | | 917,504 | | 17,152 | | 24 | Symbol values |
| **202,255,272** | | 988,680 | | 71,176 | | 24 | Decimal = 0 |

The value of decimal is initially set to 0. When the loop is terminated, the value of decimal is 202,255,272.

## Conversion from Decimal to Any Base

Conversion from a decimal value to any base can be done if we continuously divide the decimal number by the base to find the remainder and the quotient. The remainder is the face value of the next symbol; the quotient is the decimal value to be used in the next iteration. As in the case of inverse conversion, we need to have a separate algorithm to change the face value of a symbol, in the corresponding base, to the actual symbol and insert it in the string representing the converted number. We leave the detail of this subalgorithm as an exercise. Figure B.6 shows the main algorithm.

**Figure B.6** *Conversion from decimal to any base*




We can show how we can manually follow the algorithm in a few examples.

### Example B.3

Convert the decimal number 25 to its binary equivalent.

**Solution**

We continuously divide the decimal value by 2 (the base of binary system) until the quotient becomes 0. In each division we interpret the value of the remainder as the next symbol to be

inserted in the hexadecimal string. The down arrow shows the remainder; the left arrow shows the quotient. When the decimal value becomes 0, we stop. The result is the binary string $(11001)_2$.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ← | 1 | ← | 3 | ← | 6 | ← | 12 | ← | **25** | Decimal |
| | | ↓ | | ↓ | | ↓ | | ↓ | | ↓ | |
| | | 1 | | 1 | | 0 | | 0 | | 1 | Binary |

### Example B.4

Convert the decimal number 21,432 to its hexadecimal equivalent.

**Solution**

We continuously divide the decimal value by 16 (the base of hexadecimal system) until the quotient becomes 0. In each division we interpret the value of the remainder as the next symbol to be inserted in the hexadecimal string. The result is the hexadecimal string $(53B8)_{16}$.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | ← | 5 | ← | 83 | ← | 1339 | ← | **21432** | Decimal |
| | | ↓ | | ↓ | | ↓ | | ↓ | |
| | | 5 | | 3 | | B | | 8 | Hexadecimal |

### Example B.5

Convert the decimal number 73,234,122 to base 256 (IPv4 address).

**Solution**

We continuously divide the decimal value by 256 (the base) until the quotient becomes 0. In each division we interpret the value of the remainder as the next symbol to be inserted in the IPv6 address. We also insert dots as required in the dotted-decimal notation. The result is the IPv4 address 4.93.118.202.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | ← | 4 | ← | 1,117 | ← | 286,070 | ← | **73,234,122** | Decimal |
| | | ↓ | | ↓ | | ↓ | | ↓ | |
| | | 4 | • | 93 | • | 118 | • | 202 | IPv4 address |

## Other Conversions

Conversion from a nondecimal system to another nondecimal system is often easier. We can easily convert a number in binary to hexadecimal by converting a group of 4 bits into 1 hexadecimal digit. We can also convert a hexadecimal digit into a group of 4 bits. We give a few examples to show the process.

### Example B.6

Convert the binary number $(1001111101)_2$ to its equivalent in hexadecimal.

**Solution**

We create groups of 4 bits from the right. We then replace each group with its equivalent hexadecimal digit. Note that we need to add two extra 0s to the last group.

| 0010 | 0111 | 1101 | Binary |
|---|---|---|---|
| ↓ | ↓ | ↓ | |
| 2 | 7 | D | Hexadecimal |

The result is $(27D)_{16}$.

### Example B.7

Convert the hexadecimal number $(3A2B)_{16}$ to its equivalent in binary.

**Solution**

We change each hexadecimal digit to its 4-bit binary equivalent.

| 3 | A | 2 | B | Hexadecimal |
|---|---|---|---|---|
| ↓ | ↓ | ↓ | ↓ | |
| 0011 | 1010 | 0010 | 1011 | Binary |

The result is $(0011\ 1010\ 0010\ 1011)_2$.

### Example 3.8

Convert the IPv4 address 112.23.78.201 to its binary format.

**Solution**

We replace each symbol to its equivalent 8-bit binary.

| 112 | • | 23 | • | 78 | • | 201 | IPv4 address |
|---|---|---|---|---|---|---|---|
| ↓ | | ↓ | | ↓ | | ↓ | |
| 01110000 | | 00010111 | | 01001110 | | 11001001 | Binary |

The result is $(01110000\ 00010111\ 01001110\ 11001001)_2$.

APPENDIX

C

# Error Detection Codes

## C.1 INTRODUCTION

Networks must be able to transfer data with acceptable accuracy. For most applications, a system must guarantee that the data received are identical to the data transmitted. Data can become corrupted in passage. Some applications require a mechanism for detecting and, eventually, correcting errors.

### Types of Errors

Error can affect only one bit or multiple bits. The term **single-bit error** means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed. The term ***burst error*** means that two or more bits in the data unit have changed. Figure C.1 shows the effect of a single-bit or burst error on a data unit.

A burst error is more likely to occur than a single-bit error. The duration of noise is normally longer than the duration of 1 bit, which means that when noise affects data, it affects a set of bits.

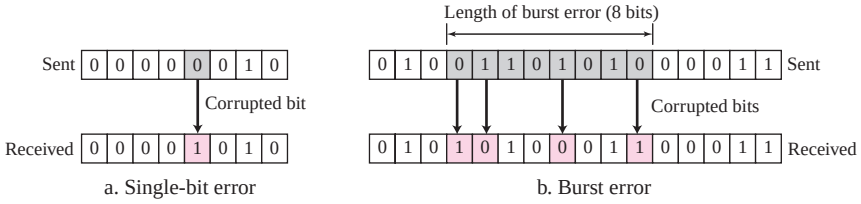
### Redundancy

The central concept in detecting or correcting errors is **redundancy.** To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect, and eventually, correct corrupted bits.

### Detection versus Correction

The correction of errors is more difficult than the detection. In **error detection,** we are looking only to see if any error has occurred. The answer is a simple yes or no. We are not even interested in the number of errors. In **error correction,** we need to know the exact number of bits that are corrupted and, more importantly, their location in the message. There are two main methods of error correction. **Forward error correction** is the process in which the receiver tries to guess the message by using

**Figure C.1** *Single-bit and burst errors*

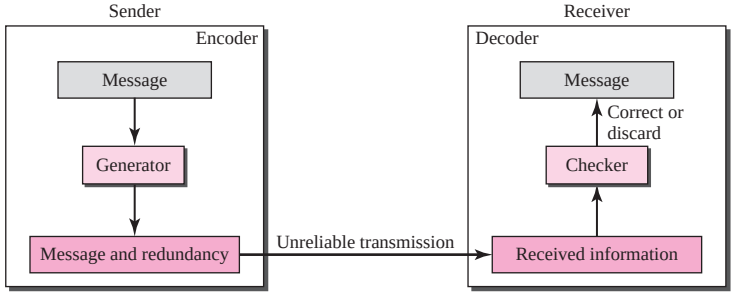



redundant bits. Correction by **retransmission** is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message. In this appendix, we discuss only error detection, assuming that the correction is achieved through retransmission.

## Coding

Redundancy is achieved through various coding schemes. The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits. The receiver checks the relationships between the two sets of bits to detect or correct the errors. The ratio of redundant bits to the data bits and the robustness of the process are important factors in any coding scheme. Figure C.2 shows the general idea of coding.

**Figure C.2** *The structure of encoder and decoder*

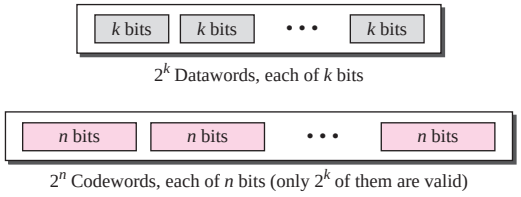



We can divide coding schemes into two broad categories: **block coding** and **convolution coding.** In this appendix, we concentrate on block coding; convolution coding is more complex and beyond the scope of this book.

## C.2 BLOCK CODING

In block coding, we divide our message into blocks, each of $k$ bits, called **datawords.** We add $r$ redundant bits to each block to make the length $n = k + r$. The resulting $n$-bit blocks are called **codewords.** How the extra $r$ bits is chosen or calculated is something we will discuss later. For the moment, it is important to know that we have a set of datawords, each of size $k$, and a set of codewords, each of size of $n$. With $k$ bits, we can create a combination of $2^k$ datawords; with $n$ bits, we can create a combination of $2^n$ codewords. Since $n > k$, the number of possible codewords is larger than the number of possible datawords. The block coding process is one-to-one; the same dataword is always encoded as the same codeword. This means that we have $2^n - 2^k$ codewords that are not used. We call these codewords invalid. Figure C.3 shows the situation.

**Figure C.3** *Datawords and codewords in block coding*




### Error Detection

How can errors be detected by using block coding? If the following two conditions are met, the receiver can detect a change in the original codeword.

1. The receiver has (or can find) a list of valid codewords.
2. The original codeword has changed to an invalid one.

The sender creates codewords out of datawords by using a generator that applies the rules and procedures of encoding (discussed later). Each codeword sent to the receiver may change during transmission. If the received codeword is the same as one of the valid codewords, the word is accepted; the corresponding dataword is extracted for use. If the received codeword is not valid, it is discarded. However, if the codeword is corrupted during transmission but the received word still matches a valid codeword, the error remains undetected.

### Hamming Distance

One of the central concepts in coding for error control is the idea of the Hamming distance. The **Hamming distance** between two words (of the same size) is the number of differences between the corresponding bits. We show the Hamming distance between two words $x$ and $y$ as $d(x, y)$. The Hamming distance can easily be found if we apply the XOR operation ($\oplus$) on the two words and count the number of 1s in the result.

## Minimum Hamming Distance

Although the concept of the Hamming distance is the central point in dealing with error detection and correction codes, the measurement that is used for designing a code is the minimum Hamming distance. In a set of words, the **minimum Hamming distance** is the smallest Hamming distance between all possible pairs. We use $d_{min}$ to define the minimum Hamming distance in a coding scheme. To find this value, we find the Hamming distances between all words and select the smallest one. A coding scheme $C$ is written as $C(n, k)$ with a separate expression for $d_{min}$.

### *Hamming Distance and Error*

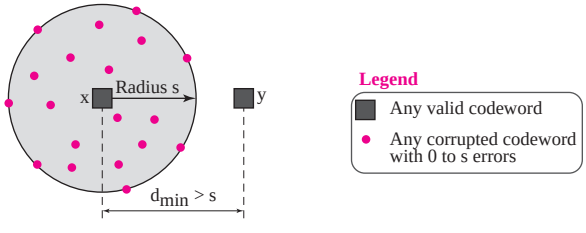
Before we explore the criteria for error detection or correction, let us discuss the relationship between the Hamming distance and errors occurring during transmission. When a codeword is corrupted during transmission, the Hamming distance between the sent and received codewords is the number of bits affected by the error. In other words, the Hamming distance between the received codeword and the sent codeword is the number of bits that are corrupted during transmission. For example, if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Hamming distance between the two is $d(00000, 01101) = 3$.

### *Minimum Distance for Error Detection*

Now let us find the minimum Hamming distance in a code if we want to be able to detect up to $s$ errors. If $s$ errors occur during transmission, the Hamming distance between the sent codeword and received codeword is $s$. If our code is to detect up to $s$ errors, the minimum distance between the valid codes must be $s + 1$, so that the received codeword does not match a valid codeword. In other words, if the minimum distance between all valid codewords is $s + 1$, the received codeword cannot be erroneously mistaken for another codeword. The error will be detected. We need to clarify a point here: Although a code with $d_{min} = s + 1$ may be able to detect more than $s$ errors in some special cases, only $s$ or fewer errors are guaranteed to be detected.

We can look at this geometrically. Let us assume that the sent codeword $x$ is at the center of a circle with radius $s$. All other received codewords that are created by 1 to $s$ errors are points inside the circle or on the perimeter of the circle. All other valid codewords must be outside the circle, as shown in Figure C.4. As the figure shows $d_{min}$ must be an integer greater than $s$; that is, $d_{min} = s + 1$.

**Figure C.4** *Geometric concept for finding $d_{min}$ in error detection*



# C.3 LINEAR BLOCK CODES

Almost all block codes used today belong to a subset called **linear block codes.** The use of nonlinear block codes for error detection and correction is not as widespread because their structure makes theoretical analysis and implementation difficult. We therefore concentrate on linear block codes.

The formal definition of linear block codes requires the knowledge of abstract algebra (particularly Galois fields), which is beyond the scope of this book. We therefore give an informal definition. For our purposes, a linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.

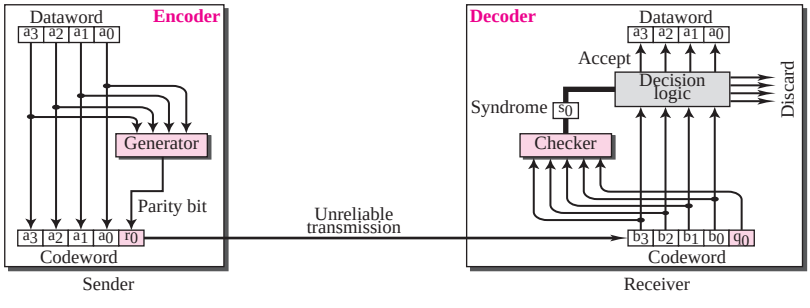
## Minimum Distance for Linear Block Codes

It is simple to find the minimum Hamming distance for a linear block code. The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

### *Simple Parity-Check Code*

Perhaps the most familiar error-detecting code is the **simple parity-check code.** In this code, a $k$-bit dataword is changed to an $n$-bit codeword where $n = k + 1$. The extra bit, called the parity bit, is selected to make the total number of 1s in the codeword even. Although some implementations specify an odd number of 1s, we discuss the even case. The minimum Hamming distance for this category is $d_{min} = 2$, which means that the code is a single-bit error-detecting code. Figure C.5 shows a possible structure of an encoder (at the sender) and a decoder (at the receiver).

**Figure C.5** *Encoder and decoder for simple parity-check code*




The encoder uses a generator that takes a copy of a 4-bit dataword ($a_0$, $a_1$, $a_2$, and $a_3$) and generates a parity bit $r_0$. The dataword bits and the **parity bit** create the 5-bit codeword. The parity bit that is added makes the number of 1s in the codeword even. This is normally done by adding the 4 bits of the dataword (modulo-2); the result is the parity bit. In other words,

$$r_0 = (a_3 + a_2 + a_1 + a_0) \text{ modulo } 2$$

If the number of 1s is even, the result is 0; if the number of 1s is odd, the result is 1. In both cases, the total number of 1s in the codeword is even.

The sender sends the codeword, which may be corrupted during transmission. The receiver receives a 5-bit word. The checker at the receiver does the same thing as the generator in the sender with one exception: The addition is done over all 5 bits. The result, which is called the **syndrome,** is just 1 bit. The syndrome is 0 when the number of 1s in the received codeword is even; otherwise, it is 1.

$$s_0 = (b_3 + b_2 + b_1 + b_0 + q_0) \text{ modulo } 2$$

The syndrome is passed to the decision logic analyzer. If the syndrome is 0, there is no error in the received codeword; the data portion of the received codeword is accepted as the dataword; if the syndrome is 1, the data portion of the received codeword is discarded. The dataword is not created.

### *Hamming Codes*

Now let us discuss a category of error-correcting codes called Hamming codes. These codes were originally designed with $d_{min} = 3$, which means that they can detect up to two errors.

First let us find the relationship between $n$ and $k$ in a Hamming code. We need to choose an integer $m \geq 3$. The values of $n$ and $k$ are then calculated from $m$ as $n = 2^m - 1$ and $k = n - m$. The number of check bits $r = m$. For example, if $m$ is 3, then $n$ is 7 and $k$ is 4. This is a Hamming code C(7, 4) with $d_{min} = 3$. It can detect at least two errors. We leave the process of creating codewords to books dedicated to error correction.

## C.4 CYCLIC CODES

Cyclic codes are special linear block codes with one extra property. In a **cyclic code,** if a codeword is cyclically shifted (rotated), the result is another codeword. For example, if 1011000 is a codeword and we cyclically left-shift it, then 0110001 is also a codeword.

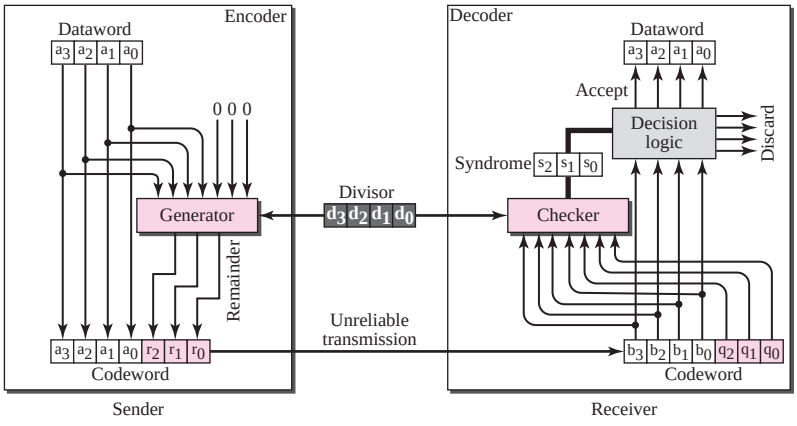
### Cyclic Redundancy Check

We can create cyclic codes to correct errors. However, the theoretical background required is beyond the scope of this book. In this section, we simply discuss a category of cyclic codes called the **cyclic redundancy check (CRC)** that is used in networks such as LANs and WANs. Figure C.6 shows one possible design for the encoder and decoder.

In the encoder, the dataword has $k$ bits (4 here); the codeword has $n$ bits (7 here). The size of the dataword is augmented by adding $n - k$ 0s to the right-hand side of the word. The $n$-bit result is fed into the generator. The generator uses a divisor of size $n - k + 1$, predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division). The quotient of the division is discarded; the remainder ($r_2 r_1 r_0$) is appended to the dataword to create the codeword.

The decoder receives the possibly corrupted codeword. A copy of all $n$ bits is fed to the checker which is a replica of the generator. The remainder produced by the checker

**Figure C.6** *CRC encoder and decoder*

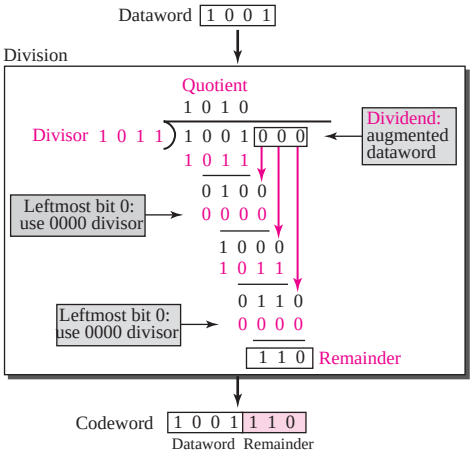



is a syndrome of $n - k$ bits, which is fed to the decision logic analyzer. The analyzer has a simple function. If the syndrome bits are all 0s, the $k$ leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the $k$ bits are discarded (error).

### *Encoder*

Let us take a closer look at the encoder. The encoder takes the dataword and augments it with $n - k$ number of 0s. It then divides the augmented dataword by the divisor, as shown in Figure C.7. The process of modulo-2 binary division is the same as the familiar

**Figure C.7** *Division in CRC encoder*



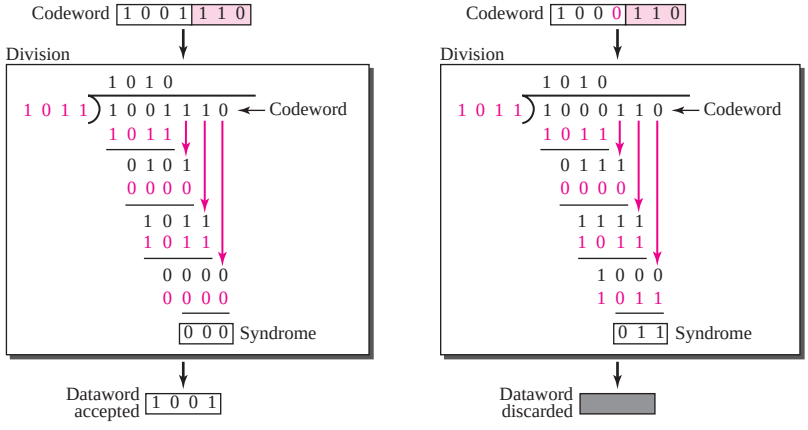
division process we use for decimal numbers. However, as mentioned at the beginning of the chapter, in this case addition and subtraction are the same. We use the XOR operation to do both. Multiplication uses the AND operation.

As in decimal division, the process is done step by step. In each step, a copy of the divisor is XORed with the 4 bits of the dividend. The result of the XOR operation (remainder) is 3 bits (in this case), which is used for the next step after 1 extra bit is pulled down to make it 4 bits long. There is one important point we need to remember in this type of division. If the leftmost bit of the dividend (or the part used in each step) is 0, the step cannot use the regular divisor; we need to use an all-0s divisor. When there are no bits left to pull down, we have a result. The 3-bit remainder forms the check bits ($r_2$, $r_1$, and $r_0$). They are appended to the dataword to create the codeword.

### *Decoder*

The codeword can be changed during transmission. The decoder does the same division process as the encoder. The remainder of the division is the syndrome. If the syndrome is all 0s, there is no error; the dataword is separated from the received codeword and accepted. Otherwise, everything is discarded. Figure C.8 shows two cases:

**Figure C.8** *Division in the CRC decoder for two cases*




The left-hand figure shows the value of syndrome when no error has occurred; the syndrome is 000. The right-hand part of the figure shows the case in which there is one single error. The syndrome is not all 0s (it is 011).

### *Divisor*

You may be wondering how the divisor 1011 is chosen. There are some criteria, but we leave the discussion to books dedicated to the error detection.

## Advantages of Cyclic Codes

Cyclic codes have a very good performance in detecting single-bit errors, double errors, an odd number of errors, and burst errors. They can easily be implemented in hardware and software. They are especially fast when implemented in hardware.

## Other Cyclic Codes

The cyclic codes we have discussed in this section are very simple. The check bits and syndromes can be calculated by simple algebra. There are, however, more powerful codes, such as **Reed-Solomon code,** used today for both detection and correction.

APPENDIX

D

# Checksum

In Appendix C, we discussed some error-detection codes. One particular error-detection method, which is prevalent in the three upper layers of the TCP/IP protocol suite, is the checksum. We discuss this method in this appendix.

## D.1 TRADITIONAL CHECKSUM

Let us first discuss the traditional checksum that has been used in the Internet. We later show some new proposals that are different from the traditional one.

### Idea

The idea of the traditional checksum is very simple. We show this idea using a simple example.

#### Example D.1

Suppose our data is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, **36**), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.

#### *One's Complement Addition*

The previous example has one major drawback. Each number can be written as a 4-bit word (each is less than 15) except for the sum. One solution is to use **one's complement** arithmetic. In this arithmetic, we can represent unsigned numbers between 0 and $2^n - 1$ using only $n$ bits. If the number has more than $n$ bits, the extra leftmost bits need to be added to the $n$ rightmost bits (wrapping).

### Example D.2

In the previous example, the decimal number 36 in binary is $(100100)_2$. To change it to a 4-bit number we add the extra rightmost bit to the left 4 bits as show below:

$$(10)_2 + (0100)_2 = (0110)_2 \rightarrow (6)_{10}$$

Instead of sending 36 as the sum, we can send 6 as the sum (7, 11, 12, 0, 6, **6**). The receiver can add the first five numbers in one's complement arithmetic. If the result is 6, the numbers are accepted; otherwise, they are rejected.

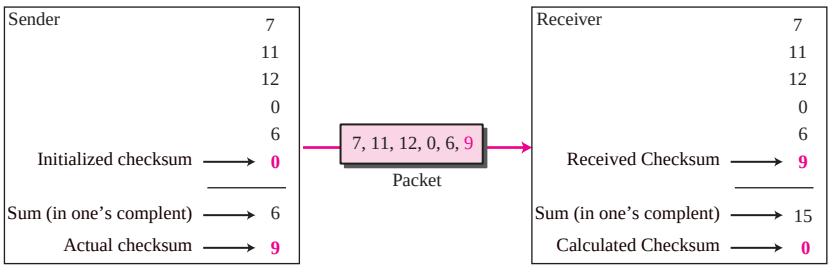
### *Checksum*

We can make the job of the receiver easier if we send the complement of the sum, called the *checksum*. In one's complement arithmetic, the complement of a number is found by completing all bits (changing all 1s to 0s and all 0s to 1s). This is the same as subtracting the number from $2^n - 1$. In one's complement arithmetic, we have two 0s: positive and negative, which are complements of each other. The positive zero has all $n$ bits set to 0; the negative zero has all bits set to 1 (it is $2^n - 1$). If we add a number with its complement, we get a negative zero (a number with all bits set to 1). When the receiver adds all five numbers (including the checksum), it gets a negative zero. The receiver can complement the result again to get a positive zero.

### Example D.3

Let us use the idea of checksum in Example D.2. The sender adds all five numbers in one's complement to get the sum = 6. The sender then complements the result to get the checksum = 9, which is 15 − 6. Note that $6 = (0110)_2$ and $9 = (1001)_2$; they are complements of each other. The sender sends the five data numbers and the checksum (7, 11, 12, 0, 6, **9**). If there is no corruption in transmission, the receiver receives (7, 11, 12, 0, 6, **9**) and adds them in one's complement to get 15. The sender complements 15 to get 0. This shows that data have not been corrupted. Figure D.1 shows the process.

### Figure D.1



## Internet Checksum

Traditionally, the Internet has used a 16-bit checksum. The sender and the receiver follow the steps depicted in Table D.1. The sender uses five steps, but the receiver uses only four.

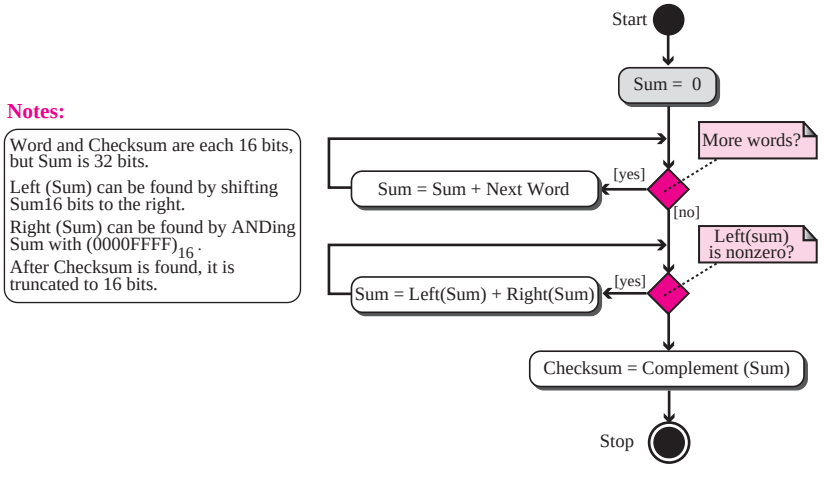
**Table D.1** *Procedure to Calculate the Traditional Checksum*

| *Sender* | *Receiver* |
|---|---|
| 1. The message is divided into 16-bit words.<br>2. The value of the checksum word is initially set to zero.<br>3. All words including the checksum are added using one's complement addition.<br>4. The sum is complemented and becomes the checksum.<br>5. The checksum is sent with the data. | 1. The message is divided into 16-bit words.<br>2. All words are added using one's complement addition.<br>3. The sum is complemented and becomes the new checksum.<br>4. If the value of checksum is 0, the message is accepted; otherwise, it is rejected. |

### *Algorithm*

We can use the flow diagram of Figure D.2 to show the algorithm for calculation of the checksum. A program in any language can be easily written based on the algorithm.

**Figure D.2** *Algorithm to calculate traditional checksum*




### *Performance*

The traditional checksum uses a small number of bits (16) to detect errors in a message of any size (sometimes thousands of bits). However, it is not as strong as the CRC in error-checking capability. For example, if the value of one word is incremented and the value of another word is decremented by the same amount, the two errors cannot be detected because the sum and checksum remain the same. Also if the values of several words are incremented but the sum and the checksum do not change, the errors are not

detected. Fletcher and Adler have proposed some weighted checksums, in which each word is multiplied by a number (its weight) that is related to its position in the text. This will eliminate the first problem we mentioned. However, the tendency in the Internet, particularly in designing new protocols, is to replace the checksum with a CRC.

## D.2 FLETCHER

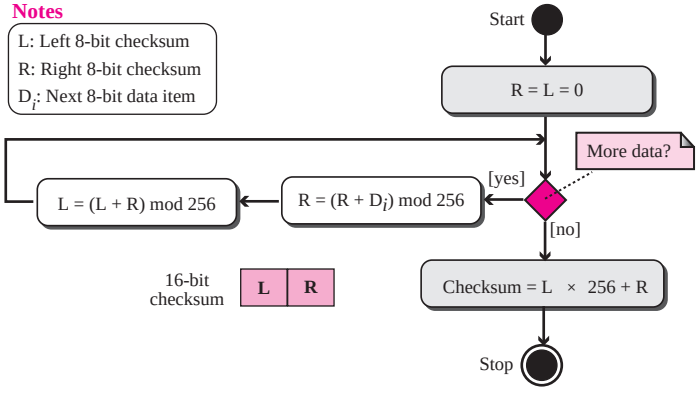
As mentioned before, there is one major problem with the traditional checksum calculation. If two 16-bit items are transposed in transmission, the checksum cannot catch this error. The reason is that the traditional checksum is not weighted: it treats each data item equally. In other words, the order of data item is immaterial to the calculation. The Fletcher checksum was devised to weight each data item according to its position.

Fletcher has proposed two algorithms: 8-bit and 16-bit. The first, 8-bit Fletcher, calculates on 8-bit data items and creates a 16-bit checksum. The second, 16-bit Fletcher, calculates on 16-bit data items and creates a 32-bit checksum.

### *Eight-Bit Fletcher*

The 8-bit Fletcher is calculated over data octets (bytes) and creates a 16-bit checksum. The calculation is done modulo 256 ($2^8$), which means the intermediate results are divided by 256 and the remainder is kept. The algorithm uses two accumulators, L and R. The first simply adds data items together; the second adds a weight to the calculation. There are many variations of the 8-bit Fletcher algorithm; we show a simple one in Figure D.3.

**Figure D.3** *Algorithm to calculate an 8-bit Fletcher checksum*




It can be proved that the accumulator L is a weighted sum of the data items. We have

$$R = D_1 + D_2 + \cdots + D_n$$

$$L = nD_1 + (n-1)D_2 + \cdots + D_n$$

If, for example, $D_1$ and $D_2$ are swapped during the transmission, the calculation of L at the receiver is different from the one done at the sender.

As an example, let us calculate the eight-bit Fletcher checksum for the string "Forouzan". We change each character to its equivalent ASCII value and calculate the values of R and L in Table D.2.

**Table D.2** *Example of an 8-bit Fletcher Checksum*

| *Byte* | $D_i$ | $R = 0$ | $L = 0$ |
|---|---|---|---|
| F | 70 | R = 0 + 70 = 70 | B = 0 + 70 = 70 |
| o | 111 | R = 70 +111 = 181 | L = 70 + 181 = 251 |
| r | 114 | R = 181 + 114 = 39 | L = 251 + 39 = 34 |
| o | 111 | R = 39 + 111 = 150 | L = 34 + 150 = 184 |
| u | 117 | R = 150 + 117 = 11 | L = 184 + 11 = 195 |
| z | 122 | R = 11 + 122 = 133 | L = 195 + 133 = 72 |
| a | 97 | R = 133 + 97 = 230 | L = 72 + 230 = 46 |
| n | 110 | R = 230 + 110 = 84 | L = 46 + 84 = 130 |
| | | | Checksum = L × **256** + R = 33,364 |

The 16-bit checksum in this case is $(8254)_{16}$. Note that the checksum is actually the concatenation of $L = (82)_{16}$ and $R = (54)_{16}$. In other words, when R and L are calculated, L goes to the leftmost byte and R to the rightmost byte.

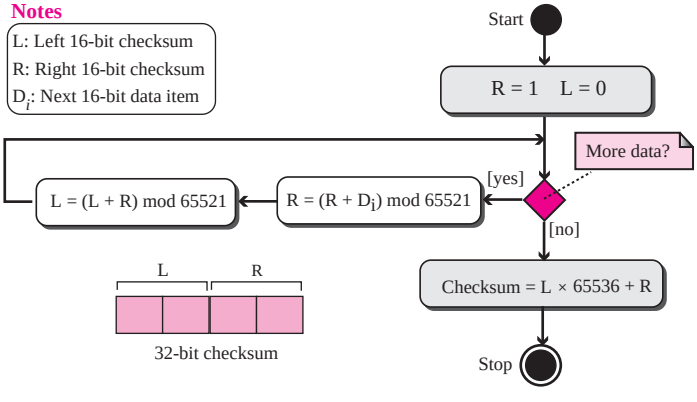
### *Sixteen-Bit Fletcher*

The 16-bit Fletcher checksum is calculated over 16-bit data items and creates a 32-bit checksum. The calculation is done modulo 65536.

## D.3 ADLER

The Adler checksum is a 32-bit checksum. Figure D.4 shows a simple algorithm in flowchart form.

**Figure D.4** *Algorithm for calculating Adler checksum*



It is similar to the 16-bit Fletcher with three differences. First, calculation is done on single bytes instead of 2 bytes at a time. Second, the modulus is a prime number (65,521) instead of 65,536. Third, L is initialized to 1 instead of 0. It has been proved that a prime modulo has a better detecting capability in some combinations of data.

Let us calculate the Adler checksum for the string "Forouzan". We change each character to its equivalent ASCII value and calculate the values of R and L in Table D.3. The 32-bit checksum in this case is $(0E8A0355)_{16}$. Note that the checksum is actually the concatenation of $L = (0E8A)_{16}$ and $R = (0355)_{16}$.

**Table D.3** *Example of Adler Checksum*

| *Byte* | $D_i$ | $R = 1$ | $L = 0$ |
|---|---|---|---|
| F | 70 | R = 1 + 70 = 71 | L = 0 + 71 = 71 |
| o | 111 | R = 71 +111 = 182 | L = 71 + 182 = 253 |
| r | 114 | R = 182 + 114 = 296 | L = 253 + 296 = 549 |
| o | 111 | R = 296 + 111 = 407 | L = 549 + 407 = 956 |
| u | 117 | R = 407 + 117 = 524 | L = 956 + 524 = 1,480 |
| z | 122 | R = 524 + 122 = 646 | L = 1480 + 646 = 2,126 |
| a | 97 | R = 646 + 97 = 743 | L = 2126 + 743= 2,869 |
| n | 110 | R = 743 + 110 = 853 | L = 2869 + 853 = 3,722 |
| Checksum = 3,722 × **65,536** + 853 = 243,925,845 | | | |

APPENDIX

E

# HTML, XHTML, XML, and XSL

## E.1 HTML

**Hypertext Markup Language (HTML)** is a language for creating Web pages. The term *markup language* comes from the book publishing industry. Before a book is typeset and printed, a copy editor reads the manuscript and puts marks on it. These marks tell the compositor how to format the text. For example, if the copy editor wants part of a line to be printed in boldface, he or she draws a wavy line under that part. In the same way, data for a Web page are formatted for interpretation by a browser.

### Tags

An HTML document is made of text and some commands that define how the text should be formatted or interpreted by the browser that displays the contents of the document. A command in HTML is called a **tag.** When a browser encounters a tag, it removes the tag and formats or interprets the text that follows. Since different parts of a document may need to be formatted or interpreted differently, the tags are designed as pairs: *beginning tag* and *ending tag.* However, some tags may have no explicit ending symbol because the ending is implicit from the text. The following shows the general format of a pair of tags.

```
<tagName>                …                </tagName>
```

The <tagName> is the beginning (opening) tag; the </tagName> is the ending (closing) tag. Between them we can have a single-line or multiline text. The tagName can be either in lowercase or uppercase, but we use lowercase in this appendix.

In addition to a name, a tag can have a set of *attributes* and the corresponding *values*. An attribute and a value define more information, as we discuss later.

```
<tagName  attribute = value  attribute = value … >
```

In this appendix, we discuss a limited number of tags to give an idea about HTML. For a comprehensive list of tags see some books dedicated to HTML.

### *Document Tag*

A web page in HTML is called a document. A **document** in HTML starts and ends with predefined tags that distinguish an HTML document from other types of documents. The beginning tag is <html>; the ending tag is </html>. Between these two tags, we can have text and other tags that interpret the text. The following shows the format of a document using tags.

```
<html>
                …
</html>
```

Although there is no need to indent the contents of the document, we often do so for clarity.

### *Head and Body Tags*

An HTML document is made of two parts: the head and the body. The head of the document normally contains only the title of the documents, but there can be some other informational materials. We show only the title in this section. Note that the title is not displayed by the browser, it has only informational value. The body of the document embeds text, image, link, and other information in an HTML document. The following shows an HTML document including head and body.

```
<html>
   <head>
            <title>  TCP/IP Protocol Suite  </title>
   </head>
   <body>
            …
   </body>
</html>
```

We have used only a single-line title, but the title can span several lines. Note also that we have not used any formatting for the title because the title is not displaced by the browser.

### *Paragraph and Line Tags*

Probably the first tags we need to learn are those that separate our paragraphs in the text and organize text into lines. The paragraph tags are  and ; the line break tags are <br> and </br>. Using a pair of line breaks without any text in between create a

blank line in the display text. The following shows an example of using these tags (in the body of the document).

```
<p>
    <br> This is the first line </br>
    <br> </br>
    <br> This is the third line </br>
</p>
```

### Appearance Tags

The next set of tags allows us to make text bold or italic. The bold tags are the pair  and ; the italic tags are  and . The following shows an example of using these tags.

```
<b> This is the first line </b>
<i> This is the third line </i>
```

The first line will be displayed in bold; the second line will be displayed in italic.

### Heading Tags

We can define up to six levels of heading in HTML, although only a few levels are normally used. The heading tags are <h*n*> and </h*n*>, in which *n* defines the level (1 through 6). For example, the following shows how we can define two header levels in a document.

```
<br> <h1>  TCP/IP Protocol Suite  </h1> </br>
<br> <h2>  A Book by B. Forouzan  </h2> </br>
```

The first line will be displayed using a larger font; the second line will be displayed using a smaller font.

### List Tags

In HTML, we can define two types of lists: unordered list and ordered list. An unordered list tags are <ul> and </ul>. Each item in the list then needs the tag <li> and </li>. The items in the list are normally bulleted and the browser inserts a line break at the end of each item. An ordered list tags are <ol> and </ol>. Each item in the list then needs the tag <li> and </li>. The items in the list then are numbered and the browser inserts a line break at the end of each item. The following shows one example of an unordered and one example of ordered list (in two separate documents).

```
<ul>
   <li> CIS 011 </li>
   <li> CIS 015 </li>
   <li> CIS 051 </li>
</ul>
```

```
<ol>
   <li> CIS 011 </li>
   <li> CIS 015 </li>
   <li> CIS 051 </li>
</ol>
```

### Image Tag

We can also include images (graphics) in a document. However, an image is not directly embedded in the document; the image tag just includes a reference to the place where the image is stored. The browser is responsible to go to the location defined in the tag, get the image, and display it in the place where the tag is located. The format of the image tag is shown below:

```
<img attribute = value attribute = value … >
```

Several attributes can be used in the image tag including:

- **src** The src (source) attribute defines the address (URL) of the location where the image can be found. The value needs to be enclosed in double quotes.
- **align** The align attribute defines how the image can be aligned with respect to the text in the document. The values can be *top*, *middle*, and *bottom*.
- **alt** The alt (alternate) attribute defines the text to replace the image if for any reason it cannot be displaced.

The following shows an example of an image tag:

```
<img src = "mypicture.gif"  alt = "Family Picture"  align = middle >
```

### Link Tag

The main idea behind HTML is to provide hypertext links. The links in an HTML document allows the user to navigate from one document, somewhere in the world, to another document, possibly somewhere else in the world. The tags are <a> and </a>. A link to another document is made of two parts: the URL of the document and the anchor. The anchor defines the text (underscore or colored) or an image that appears in the document when it is viewed by the user. The user clicks on the anchor to move to the new document. The following shows the format of the link tag.

```
<a href = value>  anchor  </a>
```

The following shows an example of link tag:

```
To find the site of this book, please go to:
<a href = "http://www.mhhe.com/engcs/compsci/forouzan/">  McGraw-Hill  </a>
```

Only the text McGraw-Hill will be shown after the colon. When the user clicks on the anchor, the browser goes to the McGraw-Hill site and displays what is there.

### Form and Input Tags

HTML 2.0 has added the idea of **forms** to the previous HTML versions. In HTML 1.0, a user can download an HTML document and browse through it. To be used as commercial tool, to let customers browse through the document and then do shopping if they wish, the forms were introduced. Forms are a set of boxes (buttons) that the user

can fill and send the information to the website. The following shows an example, using form and input tags.

```
<form action = …   method = POST >
      <input name =  “user”   size = …>
      <input name =  “address”   size = …>
      …
</form>
```

### *Other Tags*

There are many other tags that we leave for the books dedicated to HTML.

## XHTML

The Extended HTML (XHTML) is the new version of the HTML (after version 4.0) that is close to XML and XSL (defined later). In general, XHTML is similar to HTML, but more restricted in using the rules of the language. In particular, there are some changes in XHTML, which are briefly discussed below:

- All tags and attributes must be in lowercase.
- The closing tag is required. If the tag in HTML does not have a closing counterpart, in XHTML we need to insert a slash before the greater-than symbol. For example, the image tag in XHTML is <image …/>.
- Attributes must be enclosed in quotes (strings or numbers).
- Tags must be nested properly.
- Every XHTML document must have a document type as defined for XML and XSL (discussed below).

# E.2 XML AND XSL

HTML uses predefined tags to format and interpret a document. HTML, however, does not provide data structure and data representation as has been defined by many programming languages like C. A program written in a programming language like C can preform two separate tasks:

1. We can define a data structure such as an array or record in the program and initialize it to appropriate values. For example, we can define a record of students with fields defined as name, id, birthday, and so on.
2. We can use the printing and formatting functions such as *printf* function to format and print the contents of the defined record.

The interesting point is that these two tasks can be independent of each other. We can change the values stored in the record without changing the printing format. We can also change the printing format to print the same record in a different format. The duties of these two tasks have been assigned to XML and XSL.

### *Extensible Markup Language (XML)*

XML is a language that allows a user to define a representation of data or data structure and assign values to each part (field) in the structure. In other words, XML is a customized HTML in which users can define their own tags, such as <name>, <id>, and so on. The only restriction is that the user needs to follow the rules defined in XML. For example, the following shows how we can define a student record with three fields: name, id, and birthday.

```
<?xml version="1.0"?>
    <student>
        <name> George Brown </name>
        <id> 2345 </id>
        <birthday> 12-08-82 </birthday>
    </student>
```

### *Extensible Style Language (XSL)*

The data defined and initialized to values in an XML document need another language, a style language, to define how the data should be displayed. One of the ways to accomplish this is to use XSL. XSL uses formatting statements and even repeating statements to define how to display data defined in an XML document. In other words, XSL is actually an *HTML document*, but the style is applied to an XML document. We leave the details of XSL format to the books dedicated to website design.

APPENDIX

F

# *Client-Server Programming in Java*

In this appendix, we touch on client-server programming in Java using *the socket interface*. Java offers network programming to access both iterative and concurrent servers. Although writing server programs for iterative servers in Java is straight forward, writing concurrent server programs are more involved because they need Java threads. In this appendix, we only concentrate on iterative programming, both for UDP and TCP. We leave programming for concurrent servers to dedicated books on network programming in Java.

In the first section of this appendix, we give two simple programs: echo client and echo server using the services of UDP. In the second section of the appendix, we also give two simple programs: echo client and echo server using the service of TCP.

To allow server programming, Java uses several classes. Some of these classes are particularly designed to be used with UDP; other classes can be used only with TCP. The DatagramSocket class, with several methods is used to create socket objects for UDP. The ServerSocket class, with several methods, is used to create socket objects for TCP. UDP also uses the class DatagramPacket, which facilitates the creation of datagram packets.

We also have used several other input/output classes, stream classes, and buffered classes that are familiar to Java programmers.

## F.1 UDP PROGRAMS

We first concentrate on UDP client and server program. Table F.1 shows the program for the UDP echo server. It is the Java version of the program in Table 17.1 (Chapter 17). We use DatagramSocket class in line 11 to create a socket. We then use DatagramPacket class in lines 15 and 16 to create a datagram packet. The blocking *receive* method in line 17 blocks the execution of the program until a packet arrives from the client. It then uses the DatagramPacket class again to create a new packet (lines 19 to 21). It finally sends the echoed data in line 22.

**Table F.1** *Echo Server Program using the Service of UDP*

```
01 // UDP echo server program
02 import java.io.*;
03 import java.net.*;
04 public class UDPEchoServer
05 {
06       public static void main (String args [ ]) throws Exception
07       {
08             byte [ ] recvBuf = new byte [256];          // Receive buffer
09             byte [ ] sendBuf = new byte [256];          // Send buffer
10             // Create server socket
11             DatagramSocket socket = new DatagramSocket (7);
12             for ( ; ; )                  // Repeat forever
13             {
14                   // Receive a datagram
15                   DatagramPacket receivePacket =
16                         new DatagramPacket (recvBuf, recvBuf.length);
17                   socket.receive (receivePacket);
18                   // Send the datagram
19                   DatagramPacket sendPacket =
20                         new DatagramPacket (sendBuf, sendBuf.length,
21                         receivePacket.getAddress (), receivePacket.getPort () );
22                   socket.send (sendPacket);
23             } // End of for loop
24       }// End of main
25 } // End of class
```

Table F.2 shows the client program (Java version of the program in Table 17.2 in Chapter 17). Line 11 creates a socket object. Line 13 shows how we find the server address using the getByName method in InetAddress class. In lines 15, 16, and 17 we read user data from the keyboard and create a send buffer. In lines 19 to 21 we send the data. The program receives echoed data in lines 22 and 23. Finally we display the echoed data using lines 26 and 27. Note that we assume that server uses port 7 for echo program.

**Table F.2** *Echo Client Program using the Service of UDP*

```
01 // UDP echo Client program
02 import java.io.*;
03 import java.net.*;
04 public class UDPEchoClient
05 {
06       public static void main (String args[ ]) throws Exception
07       {
08             byte[ ] recvBuf = new byte[256];    // Receive buffer
09             byte[ ] sendBuf = new byte[256];    // Send buffer
10             // Create client socket
11             DatagramSocket socket = new DatagramSocket();
12             // Find the server address
13             InetAddress serverAddr = InetAddress.getByName ("server name");
14             // Input String
15             BufferedReader In = new BufferedReader(new Input StreamReader(System.in));
16             String sendString = In.readLine ();
17             sendBuf = sendString.getByte();
18             // Send the datagram
19             DatagramPacket sendPacket =
20                   new DatagramPacket (sendBuf, sendBuf.length, serverAddr, serverPort);
21             socket.send (sendPacket);
22             // Receive the datagram
23             DatagramPacket recvPacket = new DatagramPacket (recvBuf, recvBuf.length);
24             socket.receive (recvPacket);
25             // Output String
26             String recvString = new String (recvPacket.getData());
27             System.out.println ("Received from server:" + recvString);
28             // Close Socket
29             socket.close ();
30       }// End of main
31 } // End of class
```

## F.2 TCP PROGRAMS

As discussed before, to make our discussion simpler, we assume an iterative TCP server (instead of a concurrent one). Table F.3 shows the iterative version of Table 17.3 discussed in Chapter 17. In line 11, we create a TCP socket. In line 15, we use one of the method in ServerSocket class to create a listen socket. Lines 17 and 18 use the InputStream and OutputStream classes to create two stream object that receive and send streams of data. Since the string sent by the client may arrive in different segments, we use a while loop to read the data in all segments before echo the data back to the client. Note that we assume that the server used port 7 for echo program.

**Table F.3** *Echo Server Program using the Service of TCP*

```
01 // TCP echo server program
02 import java.io.*;
03 import java.net.*;
04 public class TCPEchoServer
05 {
06       public static void main (String args [ ]) throws Exception
07       {
08             byte [ ] buffer = new byte [256];  // Byte buffer
09             int br = 0;   // Number of bytes read
10             // Create server socket
11             ServerSocket listenSocket = new ServerSocket (7);
12             for ( ; ; )                  // Repeat forever
13             {
14                   // Create connection socket to serve client
15                   Socket connectSocket = listenSocket.accept();
16                   // Create input and output streams to receive and send data
17                   InputStream in = connectSocket.getInputStream();
18                   OutputStream out = connectSocket.getOutputStream();
19                   // Read and write from stream
20                   while ((br = in.read(buffer)) > 0)
21                   {
22                         out.write (buffer, 0, br);
23                   } // End of while loop
24                   connectSocket.close();
25             } // End of for loop
26       }// End of main
27 } // End of class
```

Table F.4 shows the corresponding client program. We use the Socket class to create a connecting socket in line 11. Lines 13 to 15 read data from keyboard and store them in the send buffer. We send data in lines 17 and 18 and receive data back in lines 21 to 25. Lines 27 and 28 print the received data.

**Table F.4** *Echo Client Program using the Service of TCP*

```
01 // TCP echo client program
02 import java.io.*;
03 import java.net.*;
04 public class TCPEchoClient
05 {
06       public static void main (String args[ ]) throws Exception
07       {
08             byte[ ] recvBuf = new byte[256];     // Receive buffer
09             byte[ ] sendBuf = new byte[256];    // Send buffer
09             int tbr = 0;    // Total number of bytes received
09             int br;     // Number of bytes received
10             // Create  socket
11             Socket socket = new Socket("server name", 7);
12             // Input string
13             BufferedReader In = new BufferedReader(new Input StreamReader(System.in));
14             String sendString = In.readLine ();
15             sendBuf = sendString.getByte();
16             // Send data
17             OutputStream out = socket.getOutputStream ();
18             out.write (sendBuf);
19             // Receive data
20             InputStream in = socket.getIutputStream ();
21             while (tbr < sendBuf.length)
22             {
23                   br = in.read (recvBuf, tbr, sendBuf.length – tbr);
24                   tbr = tbr + br;
25             }
26             // Output the echoed string
27             String recvString = new String (recvBuf);
28             System.out.println ("Received from server:" + recvString);
29             // Close Socket
30             socket.close ();
31       }// End of main
32 } // End of class
```

APPENDIX

G

# Miscellaneous Information

## G.1 PORT NUMBERS

Table G.1 lists all port numbers we have mentioned in this book.

**Table G.1** *Ports by port number*

| *Port Number* | *UDP or TCP* | *Protocol* |
|---|---|---|
| 7 | *TCP/UDP* | ECHO |
| 13 | *UDP/TCP* | DAYTIME |
| 19 | *UDP/TCP* | CHARACTER GENERATOR |
| 20 | *TCP* | FTP-DATA |
| 21 | *TCP* | FTP-CONTROL |
| 22 | *TCP* | SSH |
| 23 | *TCP* | TELNET |
| 25 | *TCP* | SMTP |
| 37 | *UDP/TCP* | TIME |
| 67 | *UDP* | DHCP-SERVER |
| 68 | *UDP* | DHCP-CLIENT |
| 69 | *UDP* | TFTP |
| 70 | *TCP* | GOPHER |
| 79 | *TCP* | FINGER |
| 80 | *TCP* | HTTP |
| 110 | *TCP* | POP-3 |
| 111 | *UDP/TCP* | RPC |
| 143 | *TCP* | IMAP |
| 161 | *UDP* | SNMP |
| 162 | *UDP* | SNMP-TRAP |
| 179 | *TCP* | BGP |
| 443 | *TCP* | HTTPS |
| 520 | *UDP* | RIP |

## G.2 RFC

In Table G.2, we list the RFCs that are directly related to the material in this text. For more information go to the site: ***http://www.rfc-editor.org***.

**Table G.2** *RFCs for Each Protocol*

| *Protocol* | *RFC* |
|---|---|
| ARP | 826, 1029, 1166, and 1981. |
| BGP | 1654, 1771, 1773, 1997, 2439, 2918, and 3392. |
| DHCP | 3396 and 3342. |
| DNS | 1034, 1035, 1996, 2535, 3008, 3658, 3755, 3757, and 3845. |
| Forwarding | 1812, 1971, and 1980. |
| FTP | 959, 2577, and 2585. |
| HTTP | 2068 and 2109. |
| ICMP | 792, 950, 956, 957, 1016, 1122, 1256, 1305, and 1987. |
| IPMPv6 | 2461, 2894, 3122, 3810, 4443, and 4620. |
| IPv4 Addressing | 917, 927, 930, 932, 940, 950, 1122, and 1519. |
| IPv4 | 760, 781, 791, 815, 1025, 1063, 1071, 1141, 1190, 1191, 1624, and 2113. |
| IPv6 | 1365, 1550, 1678, 1680, 1682, 1683, 1686, 1688, 1726, 1752, 1826, 1883, 1884, 1886, 1887, 1955, 2080, 2373, 2452, 2463, 2465, 2466, 2472, 2492, 2545, and 2590. |
| IPv6 Addressing | 2375, 2526, 3513, 3587, 3789, and 4291. |
| IPv6 | 2460, 2461, and 2462. |
| MIB | 2578, 2579, and 2580. |
| MIME | 2046, 2047, 2048, and 2049. |
| Mobile IP | 1701, 2003, 2004, 3024, 3344, and 3775. |
| MPLS | 3031, 3032, 3036, and 3212. |
| Multicast Routing | 1584, 1585, 2117, and 2362. |
| Multimedia | 2198, 2250, 2326, 2475, 3246, 3550, and 3551. |
| OSPF | 1583 and 2328. |
| POP3 | 1939. |
| RIP | 1058 and 2453. |
| SCTP | 4820, 4895, 4960, 5043, 5061, and 5062. |
| SMTP | 2821 and 2822. |
| SNMP | 3410, 3412, 3415, and 3418. |
| SSH | 4250, 4251, 4252, 4253, 4254, and 4344. |
| TCP | 793, 813, 879, 889, 896, 1122, 1987, 1988, 1993, 1975, 2018, 2581, 3168, and 3782. |
| TELNET | 854, 855, 856, 1041, 1091, 1372, and 1572. |
| TFTP | 906, 1350, 2347, 2348, and 2349. |
| UDP | 768. |
| WWW | 1614, 1630, 1737, and 1738. |

## G.3 CONTACT ADDRESSES

Table G.3 shows the contact addresses for organizations we discussed in this book.

**Table G.3** *Contact Addresses*

| | |
|---|---|
| **ATM Forum**<br>Presidio of San Francisco<br>P.O. Box 29920<br>572B Ruger Street<br>San Francisco, CA 94129-0920<br>www.atmforum.com | **International Telecommunication Union**<br>Place des Nations CH-1211<br>Geneva 20 Switzerland<br>intwww.itu.int/home |
| **Federal Communications Commission**<br>445 12th Street S.W.<br>Washington, DC 20554<br>www.fcc.gov | **Internet Corporation for Assigned Names and Numbers (ICANN)**<br>4676 Admiralty Way, Suite 330<br>Marina del Rey, CA 90292-6601<br>www.icann.org |
| **Institute of Electrical and Electronics Engineers (IEEE)**<br>Operations Center<br>445 Hoes Lane<br>Piscataway, NJ 08854-1331<br>www.ieee.org | **Internet Engineering Task Force (IETF)**<br>E-mail: ietf-infor@ietf.org<br>www.ietf.org |
| **International Organization for Standardization (ISO)**<br>1, rue de Varembe<br>Case postale 56<br>CH-1211 Geneva 20 Switzerland<br>www.iso.org | **Internet Society (ISOC)**<br>1775 Weihle Avenue, Suite 102<br>Reston, VA 20190-5108<br>www.isoc.org |

# Glossary

**1000BASE-CX, 1000BASE-LX, 1000BASE-SX, 1000BASE-T** The IEEE 802.3 standards for Ethernet implementation with 1-Gbps data rate.

**100BASE-FX, 100BASE-T4, 100BASE-TX, 100BASE-X** The IEEE 802.3 standards for Fast Ethernet implementation with 100-Mbps data rate.

**10BASE2, 10BASE5, 10BASE-F, 10BASE-E, 10BASE-L** The IEEE 802.3 standard for Thin Ethernet with 100-Mbps data rate.

**10BASE5** The IEEE 802.3 standard for Thick Ethernet with 100-Mbps data rate.

**10GBASE-L** The IEEE 802.3 standard for Ethernet with 10-Gbps data rate.

**abstract syntax notation 1 (ASN.1)** A formal language using abstract syntax for defining the structure of a protocol data unit (PDU).

**access control** A security service that protects against unauthorized access to data.

**acknowledgment** A packet sent by the receiver to show a successful reception.

**active attack** An attack that may change the data or harm the system.

**active close** Closing a TCP connection by a client.

**active document** In the World Wide Web, a document executed at the local site using Java.

**active open** Establishment of a connection with a server by a client.

**additive cipher** The simplest monoalphabetic cipher in which each character is encrypted by adding its value with a key.

**additive increase** With slow start, a congestion avoidance strategy in which the window size is increased by just one segment instead of exponentially.

**address** An integer to identify the source or destination of a packet.

**address mask** A 32-bit integer in which the leftmost 1s define the netid.

**address resolution protocol (ARP)** In TCP/IP, a protocol for obtaining the physical address of a node when the Internet address is known.

**address space** The total number of addresses used by a protocol.

**Advanced Encryption Standard (AES)** A non-Feistel symmetric-key block cipher published by the NIST.

**Advanced Networks and Services (ANS)** The owner and operator of the Internet since 1995.

**Advanced Networks and Services Network (ANSNET)** The high-speed Internet backbone.

**Advanced Research Projects Agency (ARPA)** The government agency that funded ARPANET.

**Advanced Research Projects Agency Network (ARPANET)** The packet switching network that was funded by ARPA.

**American National Standards Institute (ANSI)** A national standards organization that defines standards in the United States.

**American Standard Code for Information Interchange (ASCII)** A character code developed by ANSI and used extensively for data communication.

**anonymous FTP** A protocol in which a remote user can access another machine without an account or password.

**anycast address** An address that lets a packet be routed to any computer in a set of computers.

**applet** A computer program for creating an active Web document. It is usually written in Java.

**application adaptation layer (AAL)** A layer in ATM protocol that carries user data.

**application layer** The seventh layer in the OSI model or the fifth layer in the TCP/IP protocol; it provides access to network resources.

**application programming interface (API)** A set of declarations, definitions, and procedures followed by programmers to write client-server programs.

**area** A collection of networks, hosts, and routers all contained within an autonomous system.

**asymmetric digital subscriber line (ADSL)** A communication technology in which the downstream data rate is higher than the upstream rate.

**asymmetric-key cryptosystem** A cryptosystem that uses two different keys for encryption and decryption: a public key for encryption and a private key for decryption.

**asymmetric-key encipherment** An encipherment using an asymmetric-key cryptosystem.

**asynchronous transfer mode (ATM)** A wide area protocol featuring high data rates and equal-sized packets (cells); ATM is suitable for transferring text, audio, and video data.

**ATM adaptation layer (AAL)** The layer in the ATM protocol that encapsulates the user data.

**ATMARP** A version of ARP protocol used over an ATM network.

**authentication** A security service that checks the identity of the party at the other end of the line.

**Authentication Header Protocol (AH)** A protocol defined by IPSec at the network layer that provides integrity service for the payload.

**autokey cipher** A stream cipher in which each subkey in the stream is the same as the previous plaintext character. The first subkey is the secret between two parties.

**autonomous system (AS)** A group of networks and routers under the authority of a single administration.

**availability** A component of information security that requires the information created and stored by an organization to be available to authorized entities.

**base 64** An encoding to represent nontextual data in MIME.

**Basic Encoding Rules (BER)** A standard that encodes data to be transferred through a network.

**Basic Service Set (BSS)** The building block of a wireless LAN as defined by the IEEE 802.11 standard.

**Bellman-Ford algorithm** An algorithm used to calculate routing tables in the distance vector routing method.

**best-effort delivery** The unreliable transmission mechanism by IP that does not guarantee message delivery.

**big-endian byte order** A format in which the most significant byte is stored or transmitted first.

**biometrics** The measurement of physiological or behavioral features that identify a person.

**bit** A binary digit with a value of 0 or 1.

**bit-oriented cipher** A cipher in which the symbols in the plaintext, the ciphertext, and the key are bits.

**block** A group of bits treated as one unit.

**block cipher** A type of cipher in which blocks of plaintext are encrypted one at a time using the same cipher key.

**bootstrap process** The booting up of a computer that requires its IP address, subnet mask, default router address, and name server address.

**Bootstrap Protocol (BOOTP)** The protocol that provides configuration information from a table (file).

**Border Gateway Protocol (BGP)** An interautonomous system routing protocol based on path vector routing.

**bridge** A network device operating at the first two layers of the OSI model with filtering and forwarding capabilities.

**broadcast address** An address that allows transmission of a message to all nodes of a network.

**broadcast/unknown server (BUS)** A server connected to an ATM switch that can multicast and broadcast frames.

**browser** An application program that displays a WWW document. A browser usually uses other Internet services to access the document.

**buffer** Memory set aside for temporary storage.

**byte** A group of 8 bits. An octet.

**caching** The storing of information in a small, fast memory used to hold data items that are being processed.

**Caesar cipher** An additive cipher with a fixed-value key used by Julius Caesar.

**carrier sense multiple access with collision avoidance (CSMA/CA)** An access method in wireless LANs that avoids collision by forcing the stations to send reservation messages when they find the channel is idle.

**carrier sense multiple access with collision detection (CSMA/CD)** An access method in which stations transmit whenever the transmission medium is available and retransmit when collision occurs.

**cell** A small, fixed-size data unit; also, in cellular telephony, a geographical area served by a cell office.

**Certification Authority (CA)** An agency such as a federal or state organization that binds a public key to an entity and issues a certificate.

**challenge-response authentication** An authentication method in which the claimant proves that she *knows* a secret without sending it.

**character-oriented cipher** A cipher in which the symbols in the plaintext, the ciphertext, and the key are characters.

**checksum** A field used for error detection. It is formed by adding bit streams using one's complement arithmetic and then complementing the result.

**cipher.** A decryption and/or encryption algorithm.

**ciphertext** The message after being encrypted.

**claimant** In entity authentication, the entity whose identity needs to be proved.

**Clark's solution** A solution to prevent the silly window syndrome. An acknowledgment is sent as soon as the data arrive, but announces a window size of zero until either there is enough space to accommodate a segment of maximum size or until half of the buffer is empty.

**classful addressing** An IPv4 addressing mechanism in which the IP address space is divided into five classes: A, B, C, D, and E. Each class occupies some part of the whole.

**classless addressing** An addressing mechanism in which the IP address space is not divided into classes.

**Classless Interdomain Routing (CIDR)** A technique to reduce the number of routing table entries when supernetting is used.

**client process** A running application program on a local site that requests service from a running application program on a remote site.

**client-server model** The model of interaction between two application programs in which a program at one end (client) requests a service from a program at the other end (server).

**client-server paradigm** A paradigm in which two computers connected by an internet; each must run a program, one to provide a service and one to request a service.

**collision** The event that occurs when two transmitters send at the same time on a channel designed for only one transmission at a time; data will be destroyed.

**colon hexadecimal notation** In IPv6, an address notation consisting of 32 hexadecimal digits, with every four digits separated by a colon.

**common gateway interface (CGI)** A standard for communication between HTTP servers and executable programs. CGI is used in creating dynamic documents.

**compression P-box** A P-box with $n$ inputs and $m$ outputs, where $n > m$.

**concurrent client** A client program running at the same time with other client programs.

**concurrent server** A server that can process many requests at the same time and share its time between many requests.

**confidentiality** A security goal that defines procedures to hide information from an unauthorized entity.

**configuration file** A file containing information needed when a computer is booted.

**congestion** Excessive network or internetwork traffic causing a general degradation of service.

**connectionless iterative server** A connectionless server that processes one request at a time.

**connectionless service** A service for data transfer without connection establishment or termination.

**connection-oriented concurrent server** A connection-oriented server that can serve many clients at the same time.

**connection-oriented protocol** A protocol for data transfer with connection establishment and termination.

**connection-oriented service** A service for data transfer involving establishment and termination of a connection.

**Consultative Committee for International Telegraphy and Telephony (CCITT)** An international standards group now known as the ITU-T.

**contiguous mask** A mask composed of a run of 1s followed by a run of 0s.

**control character** A character that conveys information about the transmission rather than the actual data.

**control traffic** Highest priority traffic, such as routing and management messages.

**Core-Based Tree (CBT)** In multicasting, a group-shared protocol that uses a center router as the root of the tree.

**cryptanalysis** The science and art of breaking codes.

**cryptographic hash function** A function that creates a much shorter output from an input. To be useful, the function must be resistant to image, preimage, and collision attacks.

**Cryptographic Message Syntax (CMS)** The syntax used in S/MIME that defines the exact encoding scheme for each content type.

**cryptography** The science and art of transforming messages to make them secure and immune to attacks.

**CSNET** A network sponsored by the National Science Foundation originally intended for universities.

**Data Encryption Standard (DES)** A symmetric-key block cipher using rounds of Feistel ciphers and standardized by NIST.

**data link layer** The second layer in the OSI model. It is responsible for node-to-node delivery.

**datagram** In packet switching, an independent data unit.

**datagram socket** A structure designed to be used with a connectionless protocol such as UDP.

**decapsulation** Removal of a header and trailer from a message.

**decryption** Descrambling of the ciphertext to create the original plaintext.

**default mask** The mask for a network that is not subnetted.

**default routing** A routing method in which a router is assigned to receive all packets with no match in the routing table.

**Defense Advanced Research Projects Agency (DARPA)** A government organization, which, under the name of ARPA, funded ARPANET and the Internet.

**Defense Data Network (DDN)** The military portion of the Internet.

**denial of service** The only attack on the availability goal that may slow down or interrupt the system.

**destination address** The address of the receiver of the data unit.

**dialog control** The technique used by the session layer to control the dialog.

**Diffie-Hellman protocol** A protocol for creating a session key without using a KDC.

**digest** A condensed version of a document.

**digital signature** A security mechanism in which the sender can electronically sign the message and the receiver can verify the message to prove that the message is indeed signed by the sender.

**Digital Signature Standard (DSS)** The digital signature standard adopted by NIST under FIPS 186.

**digital subscriber line (DSL)** A technology using existing telecommunication networks to accomplish high-speed delivery of data, voice, video, and multimedia.

**Dijkstra's algorithm** In link state routing, an algorithm that finds the shortest path to other routers.

**direct delivery** A delivery in which the final destination of the packet is a host connected to the same physical network as the deliverer.

**direct sequence spread spectrum (DSSS)** A wireless transmission method in which each bit to be sent by the sender is replaced by a sequence of bits called a chip code.

**Distance Vector Multicast Routing Protocol (DVMRP)** A protocol based on distance vector routing that handles multicast routing in conjunction with IGMP.

**distance vector routing** A routing method in which each router sends its neighbors a list of networks it can reach and the distance to that network.

**distributed database** Information stored in many locations.

**DNS server** A computer that holds information about the name space.

**domain** A subtree of the domain name space.

**domain name system (DNS)** A TCP/IP application service that converts user-friendly names to IP addresses.

**dotted-decimal notation** A notation devised to make the IP address easier to read; each byte is converted to its decimal equivalent and then set off from its neighbor by a decimal.

**dual stack** Two protocols (IPv4 and IPv6) on the same station.

**dynamic document** A Web document created by running a CGI program at the server site.

**dynamic domain name system (DDNS)** A method to update the DNS master file dynamically.

**dynamic host configuration protocol (DHCP)** An extension to BOOTP that dynamically assigns configuration information.

**dynamic mapping** A technique in which a protocol is used for address resolution.

**dynamic port** An ephemeral port; a port that is neither controlled nor registered and can be used by any process.

**dynamic routing** Routing in which the routing table entries are updated automatically by the routing protocol.

**electronic mail (e-mail)** A method of sending messages electronically based on mailbox addresses rather than a direct host-to-host exchange.

**Electronic Industries Alliance (EIA)** An organization that promotes electronics manufacturing concerns. It has developed interface standards such as EIA-232, EIA-449, and EIA-530.

**Encapsulating Security Payload (ESP)** A protocol defined by IPSec that provides privacy as well as a combination of integrity and message authentication.

**encapsulation** The technique in which a data unit from one protocol is placed within the data field portion of the data unit of another protocol.

**encipherment** See *encryption*.

**encryption** Converting a message into an unintelligible form that is unreadable unless decrypted.

**entity authentication** A technique designed to let one party prove the identity of another party.

**ephemeral port** A port number used by the client.

**error control** The detection and handling of errors in data transmission.

**Ethernet** A local area network using the CSMA/CD access method.

**extended binary coded decimal interchange code (EBCDIC)** An 8-bit character code developed and used by IBM.

**Extended Service Set (ESS)** A wireless LAN service composed of two or more BSSs with APs as defined by the IEEE 802.11 standard.

**extranet** A private network that uses the TCP/IP protocol suite that allows authorized access from outside users.

**Federal Communications Commission (FCC)** A government agency that regulates radio, television, and telecommunications.

**Feistel cipher** A class of product ciphers consisting of both invertible and noninvertible components. A Feistel cipher combines all noninvertible elements in a unit (called a mixer in this text) and uses the same unit in the encryption and decryption algorithms.

**File Transfer Protocol (FTP)** In TCP/IP, an application layer protocol that transfers files between two sites.

**finite state machine** A machine that goes through a limited number of states.

**firewall** A device (usually a router) installed between the internal network of an organization and the rest of the Internet to provide security.

**flat namespace** A method to map a name to an address in which there is no hierarchical structure.

**flooding** Saturation of a network with a message.

**flow control** A technique to control the rate of flow of frames (packets or messages).

**fork** A UNIX function that creates a child process that has exactly the same image as its parent.

**forum** An organization that tests, evaluates, and standardizes a specific new technology.

**four-way handshake** A sequence of events for connection termination consisting of four steps between the client and server.

**fragmentation** The division of a packet into smaller units to accommodate a protocol's MTU.

**frame** A group of bits representing a block of data.

**frame relay** A packet-switching specification defined for the first two layers of the OSI model. There is no network layer. Error checking is done on end-to-end basis instead of on each link.

**Frame Relay Forum** A group formed by Digital Equipment Corporation, Northern Telecom, Cisco, and StrataCom to promote the acceptance and implementation of frame relay.

**frequency hopping spread spectrum (FHSS)** A wireless transmission method in which the sender transmits at one carrier frequency for a short period of time, then hops to another carrier frequency for the same amount of time, hops again for the same amount of time, and so on.

**full-duplex Ethernet** An Ethernet implementation in which every station is connected by two separate paths to the central hub.

**fully qualified domain name (FQDN)** A domain name consisting of labels beginning with the host and going back through each level to the root node.

**gateway** A device used to connect two separate networks that use different communication protocols.

**generic domain** A subdomain in the domain name system that uses generic suffixes.

**geographical routing** A routing technique in which the entire address space is divided into blocks based on physical land masses.

**Gigabit Ethernet** Ethernet with a 1,000-Mbps data rate.

**global Internet** The Internet.

**grafting** Resumption of multicast messages.

**graph** A data structure with no hierarchy.

**group-shared tree** A multicast routing feature in which each group in the system shares the same tree.

**H.323** An ITU standard for a protocol suite to be used with IP telephony.

**half-duplex mode** A transmission mode in which communication can be two-way but not at the same time.

**handshaking** A process to establish or terminate a connection.

**hardware address** An address used by a data link layer to identify a device.

**hash function** An algorithm that creates a fixed-size digest from a variable-length message.

**hashed message authentication** Authentication using a message digest.

**hashed message authentication code (HMAC)** A standard issued by NIST (FIPS 198) for a nested MAC.

**hashing** A cryptographic technique in which a fixed-length message digest is created from a variable-length message.

**header** Control information added to the beginning of a data packet.

**hierarchical name space** A name space made of several parts, with each succeeding part becoming more and more specific.

**hierarchical routing** A routing technique in which the entire address space is divided into levels based on specific criteria.

**high bit rate digital subscriber line (HDSL)** A service similar to the T1 line that can operate at lengths up to 3.6 km.

**home address** A mobile host's permanent address on its home network.

**home agent** Usually a router attached to the home network of the mobile host that receives and sends packets (for the mobile host) to the foreign agent.

**home network** A network that is the permanent home of the mobile host.

**homepage** A unit of hypertext or hypermedia available on the Web that is the main page for an organization or an individual.

**hop count** The number of nodes along a route. It is a measurement of distance in routing algorithms.

**hop limit** The number of nodes a datagram can travel before being discarded.

**host** A station or node on a network.

**host file** A file, used when the Internet was small, that mapped host names to host addresses.

**hostid** The part of an IP address that identifies a host.

**host-specific routing** A routing method in which the full IP address of a host is given in the routing table.

**host-to-host protocol** A protocol that can deliver a packet from one physical device to another.

**hybrid network** A network with a private internet and access to the global Internet.

**hypermedia** Information containing text, pictures, graphics, and sound that is linked to other documents through pointers.

**hypertext** Information containing text that is linked to other documents through pointers.

**hypertext markup language (HTML)** The computer language for specifying the contents and format of a Web document. It allows additional text to include codes that define fonts, layouts, embedded graphics, and hypertext links.

**hypertext transfer protocol (HTTP)** An application service for retrieving a Web document.

**initial vector (IV)** A block used to initialize the first iteration in a calculation.

**Institute of Electrical and Electronics Engineers (IEEE)** A group consisting of professional engineers that has specialized societies whose committees prepare standards in members' areas of specialty.

**integrity** A security service designed to protect data from modification, insertion, deletion, and replaying.

**interactive traffic** Traffic in which interaction with the user is necessary.

**interface** The boundary between two pieces of equipment. It also refers to mechanical, electrical, and functional characteristics of the connection. In network programming, a set of procedures available to the upper layer to use the services of the lower layer.

**International Organization for Standardization (ISO)** A worldwide organization that defines and develops standards on a variety of topics.

**International Telecommunications Union–Telecommunication Standardization Sector (ITU–T)** A standards organization formerly known as the CCITT.

**internet** A collection of networks connected by internetworking devices such as routers or gateways.

**Internet** A global internet that uses the TCP/IP protocol suite.

**Internet address** A 32-bit or 128-bit network-layer address used to uniquely define the connection of a host to an internet using the TCP/IP protocol.

**Internet Architecture Board (IAB)** The technical adviser to the ISOC; oversees the continuing development of the TCP/IP protocol suite.

**Internet Assigned Numbers Authority (IANA)** A group supported by the U.S. government that was responsible for the management of Internet domain names and addresses until October 1998.

**Internet Control Message Protocol (ICMP)** A protocol in the TCP/IP protocol suite that handles error and control messages. Two versions are used today ICMPv4 and ICMPv6.

**Internet Corporation for Assigned Names and Numbers (ICANN)** A private, nonprofit corporation managed by an international board that assumed IANA operations.

**Internet draft** A working Internet document (a work in progress) with no official status and a six-month lifetime.

**Internet Engineering Steering Group (IESG)** An organization that oversees the activity of IETF.

**Internet Engineering Task Force (IETF)** A group working on the design and development of the TCP/IP protocol suite and the Internet.

**Internet Group Management Protocol (IGMP)** A protocol in the TCP/IP protocol suite that handles multicasting.

**Internet Key Exchange (IKE)** A protocol designed to create security associations in IPSec.

**Internet Mail Access Protocol (IMAP)** A complex and powerful protocol to pull e-mail messages from an e-mail server.

**Internet Network Information Center (INTERNIC)** An agency responsible for collecting and distributing information about TCP/IP protocols.

**Internet Protocol (IP)** The network-layer protocol in the TCP/IP protocol suite governing connectionless transmission across packet-switching networks. Two versions commonly in use: IPv4 and IPv6.

**Internet Protocol, next generation (IPng)** Another term for the sixth version of the Internet Protocol.

**Internet Research Task Force (IRTF)** A forum of working groups focusing on long-term research topics related to the Internet.

**Internet Security Association and Key Management Protocol (ISAKMP)** A protocol designed by the NSA that implements the exchanges defined in IKE.

**Internet service provider (ISP)** Usually, a company that provides Internet services.

**Internet Society (ISOC)** The nonprofit organization established to publicize the Internet.

**Internet standard** A thoroughly tested specification that is useful to and adhered to by those who work with the Internet. It is a formalized regulation that must be followed.

**internetworking** Connecting several networks together using internetworking devices such as routers and gateways.

**intranet** A private network that uses the TCP/IP protocol suite.

**inverse cipher** The decryption algorithm.

**IP datagram** The Internetworking Protocol data unit.

**IP Security (IPSec)** A collection of protocols designed by the IETF (Internet Engineering Task Force) to provide security for a packet carried on the Internet.

**Java** A programming language used to create active Web documents.

**jitter** A phenomenon in real-time traffic caused by gaps between consecutive packets at the receiver.

**Karn's Algorithm** An algorithm that does not include the retransmitted segments in calculation of round-trip time.

**key** A set of values that the cipher, as an algorithm, operates on.

**key ring** A set of public or private keys used in PGP.

**key-distribution center (KDC)** A trusted third party that establishes a shared secret key between two parties.

**Link Control Protocol (LCP)** A PPP protocol responsible for establishing, maintaining, configuring, and terminating links.

**link local address** An IPv6 address that is used if a LAN is to use the Internet protocols but is not connected to the Internet for security reasons.

**link state database** In link state routing, a database common to all routers and made from LSP information.

**little-endian byte order** A format in which the least significant byte is stored or transmitted first.

**local address** The part of an e-mail address that defines the name of a special file, called the user mailbox, where all of the mail received for a user is stored for retrieval by the user agent.

**local area network (LAN)** A network connecting devices inside a single building or inside buildings close to each other.

**local client program** A program run locally that requests a service from a remote server program.

**local host** The computer that a user is physically using.

**local login** Using a terminal directly connected to the computer.

**local loop** The link that connects a subscriber to the telephone central office.

**logical address** An address defined in the network layer.

**logical IP subnet (LIS)** A grouping of nodes of an ATM network in which the connection is logical, not physical.

**logical tunnel** The encapsulation of a multicast packet inside a unicast packet to enable multicast routing by non-multicast routers.

**loopback address** An address used by a host to test its internal software.

**magic cookie** In DHCP, the number in the format of an IP address with the value of 99.130.83.99; indicates that options are present.

**mail access protocol** A protocol used by the remote user agent to access the mailbox and obtain the mail.

**mail transfer agent (MTA)** An SMTP component that transfers the mail across the Internet.

**Management Information Base (MIB)** The database used by SNMP that holds the information necessary for management of a network.

**mapped address** An IPv6 address used when a computer that has migrated to IPv6 wants to send a packet to a computer still using IPv4.

**mask** For IPv4, a 32-bit binary number that gives the first address in the block (the network address) when ANDed with an address in the block.

**masking** A process that extracts the address of the physical network from an IP address.

**master secret** In SSL, a 48-byte secret created from the *pre-master secret*.

**maturity level** The phases through which an RFC goes.

**maximum transfer unit (MTU)** The largest size data unit a specific network can handle.

**message authentication** Proving the authenticity of a sender in a connectionless communication.

**message authentication code (MAC)** An MDC that includes a secret between two parties.

**message digest** The fixed-length string created from applying a hash function to a message.

**Message Digest (MD)** A set of several hash algorithms designed by Ron Rivest and referred to as MD2, MD4, and MD5.

**metric** A cost assigned for passing through a network.

**metropolitan area network (MAN)** A network that can span a geographical area the size of a city.

**Military Network (MILNET)** A network for military use that was originally part of ARPANET.

**mobile host** A host that can move from one network to another.

**modern block cipher** A symmetric-key cipher in which each *n*-bit block of plaintext is encrypted to an *n*-bit block of ciphertext using the same key.

**modern stream cipher** A symmetric-key cipher in which encryption and decryption are done *r* bits at a time using a stream of keys.

**monoalphabetic cipher** A substitution cipher in which a symbol in the plaintext is always changed to the same symbol in the ciphertext, regardless of its position in the text.

**monoalphabetic substitution cipher** A cipher in which the key is a mapping between each plaintext character and the corresponding ciphertext character.

**multicast address** An address used for multicasting.

**multicast backbone (MBONE)** A set of internet routers supporting multicasting through the use of tunneling.

**Multicast Open Shortest Path First (MOSPF)** A multicast protocol that uses multicast link state routing to create a source-based least-cost tree.

**multicast router** A router with a list of loyal members related to each router interface that distributes multicast packets.

**multicast routing** Moving a multicast packet to its destinations.

**multicasting** A transmission method that allows copies of a single packet to be sent to a selected group of receivers.

**multihomed device** A device connected to more than one network.

**multimedia traffic** Traffic consisting of data, video, and audio.

**multiple unicasting** Sending multiple copies of a message, each with a different unicast destination address, from one source.

**multiplexing** The process of combining signals from multiple sources for transmission across a single data link.

**multiplicative decrease** A congestion avoidance technique in which the threshold is set to half of the last congestion window size, and the congestion window size starts from one again.

**Multipurpose Internet Mail Extension (MIME)** A supplement to SMTP that allows non-ASCII data to be sent through SMTP.

**multistation access unit (MAU)** In token ring, a device that houses individual automatic switches.

**Nagle's algorithm** An algorithm that attempts to prevent silly window syndrome at the sender's site; both the rate of data production and the network speed are taken into account.

**National Institute of Standards and Technology (NIST)** An agency in the U.S. government that develops standards and technology.

**National Science Foundation (NSF)** A government agency responsible for Internet funding.

**National Science Foundation Network (NSFNET)** A backbone funded by NSF.

**National Security Agency (NSA)** A U.S. intelligence-gathering security agency.

**national service provider (NSP)** A backbone network created and maintained by a specialized company.

**netid** The part of an IP address that identifies the network.

**Network Access Point (NAP)** A complex switching station that connects backbone networks.

**network address** An address that identifies a network to the rest of the Internet; it is the first address in a block.

**network address translation (NAT)** A technology that allows a private network to use a set of private addresses for internal communication and a set of global Internet addresses for external communication.

**network byte order** Same as big-endian byte order.

**Network Control Protocol (NCP)** In PPP, a set of control protocols that allows the encapsulation of data coming from network layer protocols.

**network file system (NFS)** A TCP/IP application protocol that allows a user to access and manipulate remote file systems as if they were local. It uses the services of Remote Procedure Call Protocol.

**Network Information Center (NIC)** An agency responsible for collecting and distributing information about TCP/IP protocols.

**network interface card (NIC)** An electronic device, internal or external to a station, that contains circuitry to enable the station to be connected to the network.

**network layer** The third layer in the OSI model (or TCP/IP protocol suite), responsible for the delivery of a packet to the final destination.

**Network Virtual Terminal (NVT)** A TCP/IP application protocol that allows remote login.

**network-specific routing** Routing in which all hosts on a network share one entry in the routing table.

**network-to-network interface (NNI)** In ATM, the interface between two networks.

**next-hop address** The address of the first router to which the packet is delivered.

**next-hop routing** A routing method in which only the address of the next hop is listed in the routing table instead of a complete list of the stops the packet must make.

**noise** Random electrical signals that can be picked up by the transmission medium and result in degradation or distortion of the data.

**noncontiguous mask** A mask composed of a series of bits that is not a string of 1s followed by a string of 0s, but a mixture of 0s and 1s.

**non-Feistel cipher** A product cipher that uses only invertible components.

**nonpersistent connection** A connection in which one TCP connection is made for each request/response.

**nonrepudiation** A security aspect in which a receiver must be able to prove that a received message came from a specific sender.

**Oakley** A key-exchange protocol developed by Hilarie Orman; it is an improved Diffie-Hellman method.

**one's complement** A representation of binary numbers in which the complement of a number is found by complementing all bits.

**one-time pad** A cipher invented by Vernam in which the key is a random sequence of symbols having the same length as the plaintext.

**one-time password** A password that is used only once.

**open shortest path first (OSPF)** An interior routing protocol based on link state routing.

**open systems interconnection (OSI)** A seven-layer model for data communication defined by ISO.

**out-of-band signaling** A method of signaling in which control data and user data travel on different channels.

**packet** Synonym for data unit, mostly used in the network layer.

**Packet Internet Groper (PING)** An application program to determine the reachability of a destination using an ICMP echo request and reply.

**packet-filter firewall** A firewall that forwards or blocks packets based on the information in the network layer and transport layer headers.

**page** A unit of hypertext or hypermedia available on the Web.

**partially qualified domain name (PQDN)** A domain name that does not include all the levels between the host and the root node.

**passive attack** A type of attack in which the attacker's goal is to obtain information; the attack does not modify data or harm the system.

**passive open** The state of a server as it waits for incoming requests from a client.

**password-based authentication** The simplest and oldest method of entity authentication, in which a password is used to identify the claimant.

**Path MTU Discovery technique** An IPv6 method to find the smallest MTU supported by any network on a path.

**path vector routing** A routing method on which BGP is based; in this method, the ASs through which a packet must pass are explicitly listed.

**P-box** A component in a modern block cipher that transposes bits.

**peer-to-peer process** A process on a sending and a receiving machine that communicates at a given layer.

**peer-to-peer paradigm** A paradigm in which two peer computers can communicate with each other to exchange services.

**persistent connection** A connection in which the server leaves the connection open for more requests after sending a response.

**physical address** The address of a device used at the data link layer (MAC address).

**physical layer** The first layer of the OSI model, responsible for the mechanical and electrical specifications of the medium.

**physical topology** The manner in which devices are connected in a network.

**piggybacking** The inclusion of acknowledgment on a data frame.

**plaintext** The message before encryption or after decryption.

**playback buffer** A buffer that stores the data until they are ready to be played.

**point-to-point link** A dedicated transmission link between two devices.

**Point-to-Point Protocol (PPP)** A protocol for data transfer across a serial line.

**poison reverse** A variation of split horizons. In this method, information received by the router is used to update the routing table and then passed out to all interfaces. However, a table entry that has come through one interface is set to a metric of 16 as it goes out through the same interface.

**policy routing** A path vector routing feature in which the routing tables are based on rules set by the network administrator rather than a metric.

**polyalphabetic cipher** A cipher in which each occurrence of a character may have a different substitute.

**port address** In TCP/IP protocol, an integer identifying a process (see *port number*).

**port number** An integer that defines a process running on a host.

**Post Office Protocol (POP)** A popular but simple SMTP mail access protocol.

**prefix** For a network, another name for the common part of the address range (similar to the netid).

**preimage resistance** The desired property of a cryptographic hash function in which, given a digest, it must be extremely difficult for the adversary to find any other message with the same digest.

**pre-master secret** In SSL, a secret exchanged between the client and server before calculation of the master secret.

**presentation layer** The sixth layer of the OSI model responsible for translation, encryption, authentication, and data compression.

**Pretty Good Privacy (PGP)** A protocol invented by Phil Zimmermann to provide e-mail with privacy, integrity, and authentication.

**privacy** A security aspect in which the message makes sense only to the intended receiver.

**private key** In an asymmetric-key cryptosystem, the key used for decryption. In a digital signature, the key is used for signing.

**private network** A network that is isolated from the Internet.

**process** A running application program.

**process-to-process communication** Communication between two running application programs.

**promiscuous ARP (proxy ARP)** A technique that creates a subnetting effect; one device answers ARP requests for multiple hosts.

**protocol** Rules for communication.

**Protocol Independent Multicast (PIM)** A multicasting protocol family with two members, PIM-DM and PIM-SM; both protocols are unicast-protocol dependent.

**Protocol Independent Multicast, Dense Mode (PIM-DM)** A source-based routing protocol that uses RPF and pruning/grafting strategies to handle multicasting.

**Protocol Independent Multicast, Sparse Mode (PIM-SM)** A group-shared routing protocol that is similar to CBT and uses a rendezvous point as the source of the tree.

**protocol suite** A stack or family of protocols defined for a complex communication system.

**proxy firewall** A firewall that filters a message based on the information available in the message itself (at the application layer).

**proxy server** A computer that keeps copies of responses to recent requests.

**pruning** Stopping the sending of multicast messages from an interface.

**pseudoheader** Information from the IP header used only for checksum calculation in UDP and TCP packets.

**public key** In an asymmetric-key cryptosystem, the key used for encryption. In digital signature, the key is used for verification.

**public-key encryption** A method of encryption based on a nonreversible encryption algorithm. The method uses two types of keys: The public key is known to the public; the private key (secret key) is known only to the receiver.

**public-key infrastructure (PKI)** A model for creating and distributing certificates based on X.509.

**pure ATM LAN** A LAN in which an ATM switch is used to connect the stations in a LAN, in the same way stations are connected to an Ethernet switch.

**pushing data** A technique in which the application program on the sending site does not wait for the window to be filled. It creates a segment and sends it immediately.

**quality of service (QoS)** A combined measure dealing with loss, delay, throughput, and so on.

**queue** A waiting list.

**quoted-printable** An encoding scheme used when the data consist mostly of ASCII characters with a small non-ASCII portion. If a character is ASCII, it is sent as is. If a character is not ASCII, it is sent as three characters. The first character is the equals sign (=). The next two characters are the hexadecimal representations of the byte.

**rate adaptive asymmetrical digital subscriber line (RADSL)** A DSL-based technology that features different data rates depending on the type of communication.

**raw socket** A structure designed for protocols that directly use the services of IP and use neither stream sockets nor datagram sockets.

**read-only memory (ROM)** Permanent memory, with contents that cannot be changed.

**real-time multimedia traffic** Traffic consisting of data, audio, and video that is simultaneously produced and used.

**real-time traffic** Traffic in one form that is simultaneously produced and used.

**Real-time Transport Control Protocol (RTCP)** A companion protocol to RTP with messages that control the flow and quality of data and allow the recipient to send feedback to the source or sources.

**Real-time Transport Protocol (RTP)** A protocol for real-time traffic; used in conjunction with UDP.

**regional ISP** A small ISP that is connected to one or more NSPs.

**registered port** A port number, ranging from 1,024 to 49,151, not assigned or controlled by IANA.

**registration** A phase of communication between a remote host and a mobile host in which the mobile host gives information about itself to the foreign agent.

**remote host** The computer that a user wishes to access while seated physically at another computer.

**remote login (rlogin)** The process of logging on to a remote computer from a terminal connected to a local computer.

**rendezvous router** A router that is the core or center for each multicast group; it becomes the root of the tree.

**rendezvous-point tree** A group-shared tree method in which there is one tree for each group.

**repeater** A device that extends the distance a signal can travel by regenerating the signal.

**repudiation** A type of attack on information integrity that can be launched by one of the two parties in the communication: the sender or the receiver.

**Request for Comment (RFC)** A formal Internet document concerning an Internet issue.

**requirement level** One of five RFC levels.

**reserved address** IP addresses set aside by the Internet authorities for the use of private networks. Or an IPv6 address with a reserved prefix.

**resolver** The DNS client that is used by a host that needs to map an address to a name or a name to an address.

**retransmission timer** A timer that controls the waiting time for an acknowledgment of a segment.

**Reverse Address Resolution Protocol (RARP)** A TCP/IP protocol that allows a host to find its Internet address, given its physical address.

**reverse path broadcasting (RPB)** A technique in which the router forwards only the packets that have traveled the shortest path from the source to the router.

**reverse path forwarding (RPF)** A technique in which the router forwards only the packets that have traveled the shortest path from the source to the router.

**reverse path multicasting (RPM)** A technique that adds pruning and grafting to RPB to create a multicast shortest-path tree that supports dynamic membership changes.

**ring topology** A topology in which the devices are connected in a ring. Each device on the ring receives the data unit from the previous device, regenerates it, and forwards it to the next device.

**rlogin** A remote login application designed by BSD UNIX.

**round** Each iterated section in an iterative block cipher.

**round-trip time (RTT)** The time required for a datagram to go from a source to a destination and then back again.

**router** An internetworking device operating at the first three OSI layers. A router is attached to two or more networks and forwards packets from one network to another.

**routing** The process performed by a router; finding the next hop for a datagram.

**Routing Information Protocol (RIP)** A routing protocol based on the distance vector routing algorithm.

**routing table** A table containing information a router needs to route packets. The information may include the network address, the cost, the address of the next hop, and so on.

**RSA cryptosystem** The most common public-key algorithm, devised by Rivest, Shamir, and Adleman.

**RSA signature scheme** A digital signature scheme that is based on the RSA cryptosystem, but changes the roles of the private and public keys. The sender uses her own private key to sign the document, and the receiver uses the sender's public key to verify it.

**salting** A method of improving password-based authentication in which a random string, called the salt, is concatenated to the password.

**S-box** A component in a block cipher that substitutes the bits in the input with new bits in the output.

**secret-key encryption** A security method in which the key for encryption is the same as the key for decryption; both sender and receiver have the same key.

**Secure Hash Algorithm (SHA)** A series of hash function standards developed by NIST and published as FIPS 180. It is mostly based on MD5.

**Secure Key Exchange Mechanism (SKEME)** A protocol designed by Hugo Krawcyzk for key exchange that uses public-key encryption for entity authentication.

**secure shell (SSH)** A client-server program that provides security.

**Secure Sockets Layer (SSL)** A protocol designed to provide security and compression services to data generated from the application layer.

**Secure/Multipurpose Internet Mail Extension (S/MIME)** An enhancement to MIME designed to provide security for the electronic mail.

**security** The protection of a network from unauthorized access, viruses, and catastrophe.

**Security Association (SA)** In IPSec, a logical relationship between two hosts.

**Security Association Database (SAD)** A two-dimensional table with each row defining a single security association (SA).

**security attacks** Attacks threatening the security goals of a system.

**security goals** The three goals of information security: confidentiality, integrity, and availability.

**Security Policy (SP)** In IPSec, a set of predefined security requirements applied to a packet when it is to be sent or when it has arrived.

**Security Policy Database (SPD)** A database of security policies (SPs).

**security services** Five services related to security goals and attacks: data confidentiality, data integrity, authentication, nonrepudiation, and access control.

**segment** The packet at the TCP layer.

**segmentation** The splitting of a message into multiple packets; usually performed at the transport layer.

**semantics** The meaning of each section of bits.

**sequence number** The number that denotes the location of a frame or packet in a message.

**session** In SSL, an association between a client and a server. After a session is established, the two parties have common information such as the session identifier, the certificate authenticating each of them (if necessary), the compression method (if needed), the cipher suite, and a master secret that is used to create keys for message authentication encryption.

**session key** A secret one-time key between two parties.

**session layer** The fifth layer of the OSI model, responsible for the establishment, management, and termination of logical connections between two end users.

**shared secret key** The key used in asymmetric-key cryptography.

**shift cipher** A type of additive cipher in which the key defines shifting of characters toward the end of the alphabet.

**shortest path** The optimal path from the source to the destination.

**silly window syndrome** A situation in which a small window size is advertised by the receiver and a small segment sent by the sender.

**Simple Mail Transfer Protocol (SMTP)** The TCP/IP protocol defining electronic mail service on the Internet.

**Simple Network Management Protocol (SNMP)** The TCP/IP protocol that specifies the process of management in the Internet.

**simplex mode** A transmission mode in which communication is one way.

**site local address** An IPv6 address used if a site having several networks uses the Internet protocols but is not connected to the Internet for security reasons.

**slash notation** A shorthand method to indicate the number of 1s in the mask.

**sliding window protocol** A protocol that allows several data units to be in transition before receiving an acknowledgment.

**slow convergence** A RIP shortcoming apparent when a change somewhere in an internet propagates very slowly through the rest of the internet.

**slow start** A congestion-control method in which the congestion window size increases exponentially at first.

**snooping** Unauthorized access to confidential information. An attack on the confidentiality goal in information security.

**socket** An end point for a process; two sockets are needed for communication.

**socket address** A structure holding an IP address and a port number.

**socket interface** An API based on UNIX that defines a set of system calls (procedures) that are an extension of system calls used in UNIX to access files.

**something inherent** A characteristic of the claimant, such as conventional signatures, fingerprints, voice, facial characteristics, retinal pattern, and handwriting, used for entity authentication.

**something known** A secret known only by the claimant that can be checked by the verifier in entity authentication.

**something possessed** Something belonging to the claimant that can prove the claimant's identity, such as a passport, a driver's license, an identification card, a credit card, or a smart card.

**sorcerer's apprentice bug** A TFTP problem for a packet that is not lost, but delayed in which every succeeding block is sent twice and every succeeding acknowledgment is received twice.

**source quench** A method, used in ICMP for flow control, in which the source is advised to slow down or stop the sending of datagrams because of congestion.

**source-based tree** A tree used for multicasting by multicasting protocols in which a single tree is made for each combination of source and group.

**source-to-destination delivery** The transmission of a message from the original sender to the intended recipient.

**spanning tree** A tree with the source as the root and group members as leaves; a tree that connects all of the nodes.

**split horizon** A method to improve RIP stability in which the router selectively chooses the interface from which updating information is sent.

**split operation** An operation in a block cipher that splits a block in the middle, creating two equal-length blocks.

**spoofing** See *masquerading*.

**spread spectrum** A wireless transmission technique that requires a bandwidth several times the original bandwidth.

**standard** A basis or model to which everyone has agreed.

**star topology** A topology in which all stations are attached to a central device (hub).

**state** In AES, a unit of data in intermediate stages consists of a matrix of 16 bytes.

**state transition diagram** A diagram to illustrate the states of a finite state machine.

**static document** On the World Wide Web, a fixed-content document that is created and stored in a server.

**stationary host** A host that remains attached to one network.

**station-to-station protocol** A method of creating a session key based on the Diffie-Hellman protocol that uses public-key certificates to prevent man-in-the-middle attacks.

**steganography** A security technique in which a message is concealed by covering it with something else.

**Steiner tree** A method to find the multicast tree in which the optimal tree is the one in which the sum of the costs of the links is minimum.

**stop-and-wait** A flow-control method in which each data unit must be acknowledged before the next one can be sent.

**stop-and-wait ARQ** An error-control protocol using stop-and-wait flow control.

**straight P-Boxes** A P-box with $n$ inputs and $n$ outputs.

**stream cipher** A type of cipher in which encryption and decryption are done one symbol (such as a character or a bit) at a time.

**stream socket** A structure designed to be used with a connection-oriented protocol such as TCP.

**Structure of Management Information (SMI)** In SNMP, a component used in network management.

**stub link** A network that is connected to only one router.

**subnet address** The network address of a subnet.

**subnet mask** The mask for a subnet.

**subnetting** Dividing a network into smaller units.

**subnetwork** A part of a network.

**substitution cipher** A cipher that replaces one symbol with another.

**suffix** For a network, the varying part (similar to the hostid) of the address. In DNS, a string used by an organization to define its host or resources.

**supernet** A network formed from two or more smaller networks.

**supernet mask** The mask for a supernet.

**supernetting** The combining of several class C blocks to create a larger range of addresses.

**switch** A device connecting multiple communication lines together.

**switched Ethernet** An Ethernet in which a switch, replacing the hub, can direct a transmission to its destination.

**switched virtual circuit (SVC)** A virtual circuit transmission method in which a virtual circuit is created and in existence only for the duration of the exchange.

**symmetric digital subscriber line (SDSL)** A DSL-based technology similar to HDSL, but using only one single twisted-pair cable.

**symmetric-key cryptosystem** A cryptosystem in which a single secret key is used for both encryption and decryption, sometimes called secret-key cryptosystem.

**symmetric-key encipherment** An encipherment using a symmetric-key cryptosystem.

**synchronous digital hierarchy (SDH)** The ITU-T equivalent of SONET.

**Synchronous Optical Network (SONET)** A standard developed by ANSI for fiber-optic technology that can transmit high-speed data. It can be used to deliver text, audio, and video.

**syntax** The structure or format of data, meaning the order in which they are presented.

**TCP/IP protocol suite** A group of hierarchical protocols used in an internet.

**Terminal Network (TELNET)** A general purpose client-server program for remote login.

**three-way handshake** A sequence of events for connection establishment or termination consisting of the request, then the acknowledgment of the request, and then confirmation of the acknowledgment.

**ticket** An encrypted message intended for entity B, but sent to entity A for delivery.

**T-lines** A hierarchy of digital lines designed to carry speech and other signals in digital forms.

**topology** The structure of a network including physical arrangement of devices.

**traffic analysis** A type of attack on confidentiality in which the attacker obtains some information by monitoring online traffic.

**trailer** Control information appended to a data unit.

**transient link** A network with several routers attached to it.

**Transmission Control Protocol (TCP)** A transport protocol in the TCP/IP protocol suite.

**Transmission Control Protocol/Internetwork Protocol (TCP/IP)** A five-layer protocol suite that defines the exchange of transmissions across the Internet.

**transport layer** The fourth layer in the OSI model; responsible for reliable end-to-end delivery and error recovery.

**Transport Layer Security (TLS)** A security protocol at the transport level designed to provide security on the WWW. An IETF version of the SSL protocol.

**transport mode** Encryption in which a TCP segment or a UDP user datagram is first encrypted and then encapsulated in an IPv6 packet.

**transposition cipher** A cipher that transposes symbols in the plaintext to create the ciphertext.

**tree** A hierarchical data structure in which each node on a tree has one single parent, and zero or more children.

**triple DES (3DES)** A cipher that uses three instances of DES ciphers for encryption and three instances of reverse DES ciphers for decryption.

**Trivial File Transfer Protocol (TFTP)** An unreliable TCP/IP protocol for file transfer that does not require complex interaction between client and server.

**tunnel mode** A mode in IPSec that protects the entire IP packet. It takes an IP packet, including the header, applies IPSec security methods to the entire packet, and then adds a new IP header.

**tunneling** In multicasting, a process in which the multicast packet is encapsulated in a unicast packet and then sent through the network.

**unattended data traffic** Traffic in which the user is not waiting (attending) for the data.

**unicast address** An address defining one single destination.

**unicasting** The sending of a packet to just one destination.

**Uniform Resource Locator (URL)** A string of characters (address) that identifies a page on the World Wide Web.

**urgent data** In TCP/IP, data that must be delivered to the application program as quickly as possible.

**urgent pointer** A pointer to the boundary between urgent data and normal data.

**user agent (UA)** An SMTP component that prepares the message, creates the envelope, and puts the message in the envelope.

**user datagram** The name of the packet in the UDP protocol.

**User Datagram Protocol (UDP)** A connectionless TCP/IP transport layer protocol.

**user interface** The interface between the user and the application.

**variable-length subnetting** The use of different masks to create subnets on a network.

**very high bit rate digital subscriber line (VDSL)** A DSL-based technology for short distances.

**virtual circuit** A logical circuit made between the sending and receiving computer.

**virtual private network (VPN)** A technology that creates a network that is physically public, but virtually private.

**web of trust** In PGP, the key rings shared by a group of people.

**well-known port** A port number that identifies a process on the server.

**Whirlpool** A cryptosystem based on altered AES.

**Whirlpool hash function** An iterated cryptographic hash function, based on the Whirlpool cryptosystem.

**wide area network (WAN)** A network that uses a technology that can span a large geographical distance.

**word** In AES, a group of 32 bits that can be treated as a single entity, a row matrix of 4 bytes, or a column matrix of 4 bytes.

**working group** An IETF committee concentrating on a specific Internet topic.

**World Wide Web (WWW)** A multimedia Internet service that allows users to traverse the Internet by moving from one document to another via links that connect them together.

**X.25** An ITU-T standard that defines the interface between a data terminal device and a packet-switching network.

**X.509** A recommendation devised by ITU and accepted by the Internet that defines certificates in a structured way.

**zone** In DNS, what a server is responsible for or has authority over.

# References


**[Bar et al. 05]** Barrett, Daniel, J., Silverman, Richard, E., and Byrnes, Robert, G. *SSH: The Secure Shell*, Sebastopol, CA: O'Reilly, 2005.

**[Bis 05]** Bishop, Matt, *Introduction to Computer Security.* Reading, MA: Addison-Wesley, 2005.

**[Cer 89]** Cerf, V. *A History of Arpanet, The Interoperability Report.*

**[Com 06]** Comer, Douglas E. *Internetworking with TCP/IP*, vol. 1. Upper Saddle River, NJ: Prentice Hall, 2006.

**[Don & Cal 01]** Danaho, Michael J., and Calvert, Kenneth, L. *TCP/IP Sockets, version C.* San Francisco, CA: Morgan Kaufmann, 2003.

**[Dor & Har 03]** Doraswamy, H., and Harkins, D. *IPSec.* Upper Saddle River, NJ: Prentice Hall, 2003.

**[Far 04]** Frankkel, S. *Demystifying the IPSec Puzzle.* Norwood, MA: Artech House, 2001.

**[For 03]** Forouzan, B. *Local Area Networks.* New York: McGraw-Hill, 2003.

**[For 07]** Forouzan, Behrouz. *Introduction to Data Communication and Networking.* New York: McGraw-Hill, 2007.

**[For 08]** Forouzan, Behrouz. *Cryptography and Network Security.* New York: McGraw-Hill, 2008.

**[Gar & Vid 04]** Garcia, A., and Widjaja, I. *Communication Networks.* New York: McGraw-Hill, 2004.

**[Gar 01]** Garret, P. *Making, Breaking Codes.* Upper Saddle River, NJ: Prentice Hall, 2001.

**[Jen et al. 86]** Jennings, D. M., Lancaster, L. M., Fuchs, I. H., Farber, D. H., and Arison, W. R. "Computer Networking for Scientists and Engineers," Science, vol. 231.

**[Kle 04]** Kleinrock, L. *The Birth of the Internet.*

**[Koz 05]** Kozierock, Charles M. *The TCP/IP Guide.* San Francisco: No Starch Press, 2005.

**[Kur & Ros 08]** Kurose, James F., and Ross, Keith W. *Computer Networking*, 4th ed. Reading, MA: Addison-Wesley, 2008.

**[Lei et al. 98]** Leiner, B., Cerf, D., Clark, R., Kahn, L., Kleinrock, D., Lynch, J., Postel, L., Roberts, and Woolf, S., *A Brief History of the Internet.*

**[Los 04]** Loshin, Pete. *IPv6: Theory, Protocol, and Practice.* San Francisco: Morgan Kaufmann, 2004.

**[Mao 04]** Mao, W. *Modern Cryptography.* Upper Saddle River, NJ: Prentice Hall, 2004.

**[Mau & Sch 01]** Mauro, D., and Schmidt, K. *Essential SNMP.* Sebastopol, CA: O'Reilly, 2001.

**[Mir 07]** Mir, Nader F. *Computer and Communication Network*. Upper Saddle River, NJ: Prentice Hall, 2007.

**[Moy 98]** Moy, John. *OSPF.* Reading, MA: Addison-Wesley, 1998.

**[Per 00]** Perlman, Radia. *Interconnections,* 2nd ed. Reading, MA: Addison-Wesley, 2000.

**[Pet & Dav 03]** Peterson, Larry L., and Davie, Bruce S. *Computer Networks,* 3rd ed. San Francisco: Morgan Kaufmann, 2003.

**[Res 01]** Rescorla, E. *SSL and TLS*. Reading, MA: Addison-Wesley, 2001.

**[Rob & Rob 96]** Robbins, Kay A., and Robbins, Steven. *Practical UNIX Programming,* Upper Saddle River, NJ: Prentice Hall, 1996.

**[Sam et al. 99]** Sami, Iren, Amer, Paul D., and Conrad Phillip T. "The Transport Layer: Tutorial and Survey," *ACM Computing Surveys*, vol. 31, no. 4, Dec. 1999.

**[Seg 98]** Segaller, S. *A Brief History of the Internet*.

**[Sta 04]** Stallings, William. *Data and Computer Communications,* 5th ed. Upper Saddle River, NJ: Prentice Hall, 1997.

**[Sta 06]** Stallings, William. *Data and Computer Communications,* 5th ed. Upper Saddle River, NJ: Prentice Hall, 1997.

**[Ste & Xie 01]** Stewart, Randall, R. and Xie, Qiaobing. *Stream Control Transmission Protocol (STCP).* Reading, MA: Addison-Wesley, 1998.

**[Ste 94]** Stevens, W. Richard. *TCP/IP Illustrated,* vol. 1. Reading, MA: Addison-Wesley, 1994.

**[Ste 95]** Stevens, W. Richard. *TCP/IP Illustrated,* vol. 2. Reading, MA: Addison-Wesley, 1995.

**[Ste et al. 04]** Stevens, W. Richard, Fenner, Bill, and Rudoff, Andrew, M. *UNIX Network Programming: The Sockets Networking API*. Reading, MA: Addison-Wesley, 2004.

**[Sti 06]** Stinson, D. *Cryptography: Theory and Practice.* New York: Chapman & Hall/CRC, 2006.

**[Tan 03]** Tanenbaum, Andrew S. *Computer Networks,* 4th ed. Upper Saddle River, NJ: Prentice Hall, 2003.

**[Tho 00]** Thomas, S. *SSL and TLS Essentials.* New York: John Wiley & Sons, 2000.

**[Wit & Zit 01]** Wittmann, R., and Zitterbart, M. *Multicast Communication.* San Francisco: Morgan Kaufmann, 2001.

# Index

## Numerics



## A

## B

## C

## D

## E

## F

## G



## H



## I

## J



## K



## L

## M

## N

## O



## P

## Q



## R

## S

## T

## U

## V



## W



## X



## Y



## Z