

TCP/IP Protocol Suite

Fourth Edition

Behrouz A. Forouzan

Acronyms

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

ESP	Encapsulating Security Payload	ISAKMP	Internet Security Association and Key Management Protocol
ESS	extended service set	ISO	International Organization for Standardization
FCC	Federal Communications Commission	ISOC	Internet Society
FCS	frame check sequence	ISP	Internet service provider
FHSS	frequency hopping spread spectrum	ITU-T	International Telecommunications Union–Telecommunication Standardization Sector
FQDN	fully qualified domain name	IV	initial vector
FTP	File Transfer Protocol	KDC	key-distribution center
HDSL	high bit rate digital subscriber line	LAN	local area network
HMAC	hashed message authentication code	LCP	Link Control Protocol
HTML	Hypertext Markup Language	LIS	logical IP subnet
HTTP	Hypertext Transfer Protocol	LSA	link state advertisement
IAB	Internet Architecture Board	MAA	message access agent
IANA	Internet Assigned Numbers Authority	MAC	media access control or message authentication code
ICANN	Internet Corporation for Assigned Names and Numbers	MBONE	multicast backbone
ICMP	Internet Control Message Protocol	MD	Message Digest
IEEE	Institute of Electrical and Electronics Engineers	MIB	management information base
IEGS	Internet Engineering Steering Group	MILNET	Military Network
IETF	Internet Engineering Task Force	MIME	Multipurpose Internet Mail Extension
IGMP	Internet Group Management Protocol	MOSPF	Multicast Open Shortest Path First
IKE	Internet Key Exchange	MSS	maximum segment size
INTERNIC	Internet Network Information Center	MTA	message transfer agent
IP	Internet Protocol	MTU	maximum transfer unit
IPng	Internetworking Protocol, next generation	NAP	Network Access Point
IPSec	IP Security	NAT	network address translation
IRTF	Internet Research Task Force	NIC	Network Information Center
		NIC	network interface card
		NIST	National Institute of Standards and Technology
		NSA	National Security Agency

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

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

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



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



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. Copyright © 2010 by The McGraw-Hill Companies, Inc. All rights reserved. Previous editions © 2006, 2003, and 2001. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written consent of The McGraw-Hill Companies, Inc., including, but not limited to, in any network or other electronic storage or transmission, or broadcast for distance learning.

Some ancillaries, including electronic and print components, may not be available to customers outside the United States.

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*

All credits appearing on page or at the end of the book are considered to be an extension of the copyright page.

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

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

—Behrouz A. Forouzan

Brief Contents

Preface **xxxi**

Trademarks **xxxv**

Part 1 ***Introduction and Underlying Technologies*** **1**

Chapter 1 ***Introduction*** **2**

Chapter 2 ***The OSI Model and the TCP/IP Protocol Suite*** **18**

Chapter 3 ***Underlying Technologies*** **46**

Part 2 ***Network Layer*** **93**

Chapter 4 ***Introduction to Network Layer*** **94**

Chapter 5 ***IPv4 Addresses*** **114**

Chapter 6 ***Delivery and Forwarding of IP Packets*** **160**

Chapter 7 ***Internet Protocol Version 4 (IPv4)*** **186**

Chapter 8 ***Address Resolution Protocol (ARP)*** **220**

Chapter 9 ***Internet Control Message Protocol Version 4 (ICMPv4)*** **244**

Chapter 10 ***Mobile IP*** **268**

Chapter 11 ***Unicast Routing Protocols (RIP, OSPF, and BGP)*** **282**

Chapter 12 ***Multicasting and Multicast Routing Protocols*** **334**

Part 3 ***Transport Layer*** **373**

Chapter 13 ***Introduction to the Transport Layer*** **374**

Chapter 14 ***User Datagram Protocol (UDP)*** **414**

Chapter 15 ***Transmission Control Protocol (TCP)*** **432**

Chapter 16 ***Stream Control Transmission Protocol (SCTP)*** **502**

Part 4	<i>Application Layer</i>	541
Chapter 17	<i>Introduction to the Application Layer</i>	542
Chapter 18	<i>Host Configuration: DHCP</i>	568
Chapter 19	<i>Domain Name System (DNS)</i>	582
Chapter 20	<i>Remote Login: TELNET and SSH</i>	610
Chapter 21	<i>File Transfer: FTP and TFTP</i>	630
Chapter 22	<i>World Wide Web and HTTP</i>	656
Chapter 23	<i>Electronic Mail: SMTP, POP, IMAP, and MIME</i>	680
Chapter 24	<i>Network Management: SNMP</i>	706
Chapter 25	<i>Multimedia</i>	728
Part 5	<i>Next Generation</i>	767
Chapter 26	<i>IPv6 Addressing</i>	768
Chapter 27	<i>IPv6 Protocol</i>	786
Chapter 28	<i>ICMPv6</i>	800
Part 6	<i>Security</i>	815
Chapter 29	<i>Cryptography and Network Security</i>	816
Chapter 30	<i>Internet Security</i>	858
Part 7	<i>Appendices</i>	891
Appendix A	<i>Unicode</i>	892
Appendix B	<i>Positional Numbering Systems</i>	896
Appendix C	<i>Error Detection Codes</i>	904
Appendix D	<i>Checksum</i>	914
Appendix E	<i>HTML, XHTML, XML, and XSL</i>	920
Appendix F	<i>Client-Server Programming in Java</i>	926
Appendix G	<i>Miscellaneous Information</i>	932
	<i>Glossary</i>	935
	<i>References</i>	955
	<i>Index</i>	957

Contents

Preface **xxxi**

Trademarks **xxxv**

Part 1 Introduction and Underlying Technologies 1

Chapter 1 Introduction 2

1.1	A BRIEF HISTORY 3
	ARPANET 3
	Birth of the Internet 3
	Transmission Control Protocol/Internetworking Protocol (TCP/IP) 4
	MILNET 4
	CSNET 4
	NSFNET 4
	ANSNET 5
	The Internet Today 5
	World Wide Web 6
	Time Line 6
	Growth of the Internet 7
1.2	PROTOCOLS AND STANDARDS 7
	Protocols 7
	Standards 8
1.3	STANDARDS ORGANIZATIONS 8
	Standards Creation Committees 8
	Forums 10
	Regulatory Agencies 10
1.4	INTERNET STANDARDS 10
	Maturity Levels 11
	Requirement Levels 12
1.5	INTERNET ADMINISTRATION 13
	Internet Society (ISOC) 13
	Internet Architecture Board (IAB) 13
	Internet Engineering Task Force (IETF) 13
	Internet Research Task Force (IRTF) 14
	Internet Assigned Numbers Authority (IANA) and Internet Corporation for Assigned Names and Numbers (ICANN) 14
	Network Information Center (NIC) 14

1.6	FURTHER READING	14
	Books and Papers	15
	Websites	15
1.7	KEY TERMS	15
1.8	SUMMARY	15
1.9	PRACTICE SET	16
	Exercises	16
	Research Activities	17

Chapter 2 The OSI Model and the TCP/IP Protocol Suite 18

2.1	PROTOCOL LAYERS	19
	Hierarchy	20
	Services	20
2.2	THE OSI MODEL	20
	Layered Architecture	21
	Layer-to-Layer Communication	22
	Encapsulation	23
	Layers in the OSI Model	24
	Summary of OSI Layers	28
2.3	TCP/IP PROTOCOL SUITE	28
	Comparison between OSI and TCP/IP Protocol Suite	28
	Layers in the TCP/IP Protocol Suite	30
2.4	ADDRESSING	35
	Physical Addresses	35
	Logical Addresses	37
	Port Addresses	39
	Application-Specific Addresses	40
2.5	FURTHER READING	40
	Books	40
	RFCs	40
2.6	KEY TERMS	41
2.7	SUMMARY	41
2.8	PRACTICE SET	42
	Exercises	42
	Research Activities	44

Chapter 3 Underlying Technologies 46

3.1	WIRED LOCAL AREA NETWORKS	47
	IEEE Standards	47
	Frame Format	48
	Addressing	49
	Ethernet Evolution	51
	Standard Ethernet	51
	Fast Ethernet	55
	Gigabit Ethernet	56
	Ten-Gigabit Ethernet	59

3.2	WIRELESS LANS	59
	IEEE 802.11	59
	MAC Sublayer	61
	Addressing Mechanism	64
	Bluetooth	67
3.3	POINT-TO-POINT WANS	70
	56K Modems	70
	DSL Technology	71
	Cable Modem	72
	T Lines	75
	SONET	75
	PPP	76
3.4	SWITCHED WANS	77
	X.25	77
	Frame Relay	78
	ATM	78
3.5	CONNECTING DEVICES	83
	Repeaters	83
	Bridges	84
	Routers	86
3.6	FURTHER READING	88
3.7	KEY TERMS	88
3.8	SUMMARY	89
3.9	PRACTICE SET	89
	Exercises	89
	Research Activities	90

Part 2 Network Layer 93

Chapter 4 Introduction to Network Layer 94

4.1	INTRODUCTION	95
4.2	SWITCHING	96
	Circuit Switching	96
	Packet Switching	96
4.3	PACKET SWITCHING AT NETWORK LAYER	97
	Connectionless Service	97
	Connection-Oriented Service	99
4.4	NETWORK LAYER SERVICES	103
	An Example	103
	Logical Addressing	104
	Services Provided at the Source Computer	105
	Services Provided at Each Router	106
	Services Provided at the Destination Computer	107
4.5	OTHER NETWORK LAYER ISSUES	108
	Error Control	108
	Flow Control	109
	Congestion Control	110

Quality of Service	111	
Routing	111	
Security	111	
4.6	FURTHER READING	111
4.7	KEY TERMS	112
4.8	SUMMARY	112
4.9	PRACTICE SET	112
	Exercises	112

Chapter 5 IPv4 Addresses 114

5.1	INTRODUCTION	115
	Address Space	115
	Notation	115
	Range of Addresses	117
	Operations	118
5.2	CLASSFUL ADDRESSING	121
	Classes	121
	Classes and Blocks	123
	Two-Level Addressing	126
	An Example	129
	Three-Level Addressing: Subnetting	131
	Supernetting	134
5.3	CLASSLESS ADDRESSING	135
	Variable-Length Blocks	136
	Two-Level Addressing	136
	Block Allocation	141
	Subnetting	142
5.4	SPECIAL ADDRESSES	147
	Special Blocks	147
	Special Addresses in Each block	148
5.5	NAT	149
	Address Translation	150
	Translation Table	150
5.6	FURTHER READING	152
	Books	152
	RFCs	152
5.7	KEY TERMS	153
5.8	SUMMARY	153
5.9	PRACTICE SET	154
	Exercises	154

Chapter 6 Delivery and Forwarding of IP Packets 160

6.1	DELIVERY	161
	Direct Delivery	161
	Indirect Delivery	161
6.2	FORWARDING	162
	Forwarding Based on Destination Address	162
	Forwarding Based on Label	176

6.3	STRUCTURE OF A ROUTER	178
	Components	178
6.4	FURTHER READING	181
	Books	182
	RFCs	182
6.5	KEY TERMS	182
6.6	SUMMARY	182
6.7	PRACTICE SET	183
	Exercises	183
	Research Activities	184

Chapter 7 Internet Protocol Version 4 (IPv4) 186

7.1	INTRODUCTION	187
7.2	DATAGRAMS	187
7.3	FRAGMENTATION	192
	Maximum Transfer Unit (MTU)	192
	Fields Related to Fragmentation	193
7.4	OPTIONS	197
	Format	197
	Option Types	198
7.5	CHECKSUM	205
	Checksum Calculation at the Sender	205
	Checksum Calculation at the Receiver	205
	Checksum in the IP Packet	206
7.6	IP OVER ATM	207
	ATM WANs	208
	Routing the Cells	208
7.7	SECURITY	210
	Security Issues	210
	IPSec	211
7.8	IP PACKAGE	211
	Header-Adding Module	212
	Processing Module	213
	Queues	213
	Routing Table	214
	Forwarding Module	214
	MTU Table	214
	Fragmentation Module	214
	Reassembly Table	215
	Reassembly Module	215
7.9	FURTHER READING	216
	Books	216
	RFCs	217
7.10	KEY TERMS	217
7.11	SUMMARY	217
7.12	PRACTICE SET	218
	Exercises	218
	Research Activities	219

Chapter 8 Address Resolution Protocol (ARP) 220

8.1	ADDRESS MAPPING	221
	Static Mapping	221
	Dynamic Mapping	222
8.2	THE ARP PROTOCOL	222
	Packet Format	223
	Encapsulation	224
	Operation	224
	Proxy ARP	226
8.3	ATMARP	228
	Packet Format	228
	ATMARP Operation	229
	Logical IP Subnet (LIS)	232
8.4	ARP PACKAGE	233
	Cache Table	233
	Queues	235
	Output Module	235
	Input Module	236
	Cache-Control Module	237
	More Examples	238
8.5	FURTHER READING	240
	Books	240
	RFCS	240
8.6	KEY TERMS	240
8.7	SUMMARY	241
8.8	PRACTICE SET	241
	Exercises	241

Chapter 9 Internet Control Message Protocol Version 4 (ICMPv4) 244

9.1	INTRODUCTION	245
9.2	MESSAGES	246
	Message Format	246
	Error Reporting Messages	246
	Query Messages	253
	Checksum	256
9.3	DEBUGGING TOOLS	257
	Ping	257
	Traceroute	259
9.4	ICMP PACKAGE	262
	Input Module	263
	Output Module	263
9.5	FURTHER READING	264
	Books	264
	RFCS	264
9.6	KEY TERMS	264
9.7	SUMMARY	265

9.8	PRACTICE SET	265
	Exercises	265
	Research Activities	267

Chapter 10 Mobile IP 268

10.1	ADDRESSING	269
	Stationary Hosts	269
	Mobile Hosts	269
10.2	AGENTS	270
	Home Agent	271
	Foreign Agent	271
10.3	THREE PHASES	271
	Agent Discovery	271
	Registration	273
	Data Transfer	275
10.4	INEFFICIENCY IN MOBILE IP	277
	Double Crossing	277
	Triangle Routing	277
	Solution	277
10.5	FURTHER READING	278
	Books	278
	RFCs	278
10.6	KEY TERMS	278
10.7	SUMMARY	279
10.8	PRACTICE SET	279
	Exercises	279
	Research Activities	280

Chapter 11 Unicast Routing Protocols (RIP, OSPF, and BGP) 282

11.1	INTRODUCTION	283
	Cost or Metric	283
	Static versus Dynamic Routing Tables	283
	Routing Protocol	283
11.2	INTRA- AND INTER-DOMAIN ROUTING	284
11.3	DISTANCE VECTOR ROUTING	285
	Bellman-Ford Algorithm	285
	Distance Vector Routing Algorithm	287
	Count to Infinity	291
11.4	RIP	293
	RIP Message Format	294
	Requests and Responses	295
	Timers in RIP	296
	RIP Version 2	297
	Encapsulation	299
11.5	LINK STATE ROUTING	299
	Building Routing Tables	300

11.6	OSPF	304
	Areas	304
	Metric	305
	Types of Links	305
	Graphical Representation	307
	OSPF Packets	307
	Link State Update Packet	309
	Other Packets	317
	Encapsulation	320
11.7	PATH VECTOR ROUTING	320
	Reachability	321
	Routing Tables	322
11.8	BGP	323
	Types of Autonomous Systems	323
	Path Attributes	324
	BGP Sessions	324
	External and Internal BGP	324
	Types of Packets	325
	Packet Format	325
	Encapsulation	329
11.9	FURTHER READING	329
	Books	329
	RFCs	330
11.10	KEY TERMS	330
11.11	SUMMARY	330
11.12	PRACTICE SET	331
	Exercises	331
	Research Activities	333

Chapter 12 Multicasting and Multicast Routing Protocols 334

12.1	INTRODUCTION	335
	Unicasting	335
	Multicasting	336
	Broadcasting	338
12.2	MULTICAST ADDRESSES	338
	Multicast Addresses in IPv4	339
	Selecting Multicast Address	341
	Delivery of Multicast Packets at Data Link Layer	342
12.3	IGMP	343
	Group Management	344
	IGMP Messages	344
	IGMP Protocol Applied to Host	347
	IGMP Protocol Applied to Router	351
	Role of IGMP in Forwarding	352
	Variables and Timers	354
	Encapsulation	355
	Compatibility with Older Versions	355
12.4	MULTICAST ROUTING	355
	Optimal Routing: Shortest Path Trees	355

- 12.5 ROUTING PROTOCOLS 358
 - Multicast Link State Routing: MOSPF 358
 - Multicast Distance Vector 360
 - DVMRP 364
 - CBT 364
 - PIM 366
- 12.6 MBONE 367
- 12.7 FURTHER READING 368
 - Books 368
 - RFCs 368
- 12.8 KEY TERMS 368
- 12.9 SUMMARY 369
- 12.10 PRACTICE SET 369
 - Exercises 369
 - Research Activities 371

Part 3 Transport Layer 373

Chapter 13 Introduction to the Transport Layer 374

- 13.1 TRANSPORT-LAYER SERVICES 375
 - Process-to-Process Communication 375
 - Addressing: Port Numbers 375
 - Encapsulation and Decapsulation 378
 - Multiplexing and Demultiplexing 379
 - Flow Control 379
 - Error Control 382
 - Combination of Flow and Error Control 383
 - Congestion Control 385
 - Connectionless and Connection-Oriented Services 386
- 13.2 TRANSPORT-LAYER PROTOCOLS 389
 - Simple Protocol 390
 - Stop-and-Wait Protocol 391
 - Go-Back-N Protocol 395
 - Selective-Repeat Protocol 403
 - Bidirectional Protocols: Piggybacking 408
- 13.3 FURTHER READING 409
- 13.4 KEY TERMS 409
- 13.5 SUMMARY 410
- 13.6 PRACTICE SET 411
 - Exercises 411
 - Research Activities 413

Chapter 14 User Datagram Protocol (UDP) 414

- 14.1 INTRODUCTION 415
- 14.2 USER DATAGRAM 416
- 14.3 UDP SERVICES 417
 - Process-to-Process Communication 417
 - Connectionless Services 418

Flow Control	418
Error Control	418
Congestion Control	420
Encapsulation and Decapsulation	420
Queueing	421
Multiplexing and Demultiplexing	423
Comparison between UDP and Generic Simple Protocol	423
14.4	UDP APPLICATIONS 424
UDP Features	424
Typical Applications	426
14.5	UDP PACKAGE 426
Control-Block Table	426
Input Queues	426
Control-Block Module	426
Input Module	427
Output Module	428
Examples	428
14.6	FURTHER READING 430
Books	430
RFCs	430
14.7	KEY TERMS 430
14.8	SUMMARY 430
14.9	PRACTICE SET 431
Exercises	431

Chapter 15 Transmission Control Protocol (TCP) 432

15.1	TCP SERVICES 433
Process-to-Process Communication	433
Stream Delivery Service	434
Full-Duplex Communication	436
Multiplexing and Demultiplexing	436
Connection-Oriented Service	436
Reliable Service	436
15.2	TCP FEATURES 437
Numbering System	437
Flow Control	438
Error Control	438
Congestion Control	439
15.3	SEGMENT 439
Format	439
Encapsulation	441
15.4	A TCP CONNECTION 442
Connection Establishment	442
Data Transfer	444
Connection Termination	446
Connection Reset	448
15.5	STATE TRANSITION DIAGRAM 449
Scenarios	450

15.6	WINDOWS IN TCP	457
	Send Window	457
	Receive Window	458
15.7	FLOW CONTROL	459
	Opening and Closing Windows	460
	Shrinking of Windows	462
	Silly Window Syndrome	463
15.8	ERROR CONTROL	465
	Checksum	465
	Acknowledgment	465
	Retransmission	466
	Out-of-Order Segments	467
	FSMs for Data Transfer in TCP	467
	Some Scenarios	468
15.9	CONGESTION CONTROL	473
	Congestion Window	473
	Congestion Policy	474
15.10	TCP TIMERS	478
	Retransmission Timer	478
	Persistence Timer	481
	Keepalive Timer	482
	TIME-WAIT Timer	482
15.11	OPTIONS	482
15.12	TCP PACKAGE	489
	Transmission Control Blocks (TCBs)	490
	Timers	491
	Main Module	491
	Input Processing Module	495
	Output Processing Module	496
15.13	FURTHER READING	496
	Books	496
	RFCs	496
15.14	KEY TERMS	496
15.15	SUMMARY	497
15.16	PRACTICE SET	498
	Exercises	498
	Research Activities	501

Chapter 16 Stream Control Transmission Protocol (SCTP) 502

16.1	INTRODUCTION	503
16.2	SCTP SERVICES	504
	Process-to-Process Communication	504
	Multiple Streams	504
	Multihoming	505
	Full-Duplex Communication	506
	Connection-Oriented Service	506
	Reliable Service	506

16.3	SCTP FEATURES	506
	Transmission Sequence Number (TSN)	506
	Stream Identifier (SI)	506
	Stream Sequence Number (SSN)	507
	Packets	507
	Acknowledgment Number	509
	Flow Control	509
	Error Control	509
	Congestion Control	510
16.4	PACKET FORMAT	510
	General Header	510
	Chunks	511
16.5	AN SCTP ASSOCIATION	519
	Association Establishment	519
	Data Transfer	521
	Association Termination	524
	Association Abortion	524
16.6	STATE TRANSITION DIAGRAM	525
	Scenarios	526
16.7	FLOW CONTROL	529
	Receiver Site	529
	Sender Site	530
	A Scenario	530
16.8	ERROR CONTROL	531
	Receiver Site	532
	Sender Site	532
	Sending Data Chunks	534
	Generating SACK Chunks	534
16.9	CONGESTION CONTROL	535
	Congestion Control and Multihoming	535
	Explicit Congestion Notification	535
16.10	FURTHER READING	535
	Books	536
	RFCs	536
16.11	KEY TERMS	536
16.12	SUMMARY	536
16.13	PRACTICE SET	537
	Exercises	537
	Research Activities	539

Part 4 Application Layer 541

Chapter 17 Introduction to the Application Layer 542

17.1	CLIENT-SERVER PARADIGM	543
	Server	544
	Client	544
	Concurrency	544

	Socket Interfaces	546
	Communication Using UDP	554
	Communication Using TCP	558
	Predefined Client-Server Applications	564
17.2	PEER-TO-PEER PARADIGM	564
17.3	FURTHER READING	565
17.4	KEY TERMS	565
17.5	SUMMARY	565
17.6	PRACTICE SET	566
	Exercises	566

Chapter 18 Host Configuration: DHCP 568

18.1	INTRODUCTION	569
	Previous Protocols	569
	DHCP	570
18.2	DHCP OPERATION	570
	Same Network	570
	Different Networks	571
	UDP Ports	572
	Using TFTP	572
	Error Control	573
	Packet Format	573
18.3	CONFIGURATION	576
	Static Address Allocation	576
	Dynamic Address Allocation	576
	Transition States	576
	Other Issues	578
	Exchanging Messages	579
18.4	FURTHER READING	579
	Books and RFCs	579
18.5	KEY TERMS	580
18.6	SUMMARY	580
18.7	PRACTICE SET	580
	Exercises	580
	Research Activities	581

Chapter 19 Domain Name System (DNS) 582

19.1	NEED FOR DNS	583
19.2	NAME SPACE	584
	Flat Name Space	584
	Hierarchical Name Space	584
	Domain Name Space	585
	Domain	587
	Distribution of Name Space	587
19.3	DNS IN THE INTERNET	589
	Generic Domains	589
	Country Domains	590

	Inverse Domain	591
	Registrar	592
19.4	RESOLUTION	593
	Resolver	593
	Mapping Names to Addresses	593
	Mapping Addresses to Names	593
	Recursive Resolution	593
	Iterative Resolution	594
	Caching	594
19.5	DNS MESSAGES	595
	Header	596
19.6	TYPES OF RECORDS	598
	Question Record	598
	Resource Record	599
19.7	COMPRESSION	600
19.8	ENCAPSULATION	604
19.9	REGISTRARS	604
19.10	DDNS	604
19.11	SECURITY OF DNS	605
19.12	FURTHER READING	605
	Books	606
	RFCs	606
19.13	KEY TERMS	606
19.14	SUMMARY	606
19.15	PRACTICE SET	607
	Exercises	607
	Research Activities	608

Chapter 20 Remote Login: TELNET and SSH 610

20.1	TELNET	611
	Concepts	611
	Time-Sharing Environment	611
	Network Virtual Terminal (NVT)	613
	Embedding	614
	Options	615
	Symmetry	618
	Suboption Negotiation	618
	Controlling the Server	618
	Out-of-Band Signaling	620
	Escape Character	620
	Modes of Operation	621
	User Interface	623
	Security Issue	624
20.2	SECURE SHELL (SSH)	624
	Versions	624
	Components	624
	Port Forwarding	625
	Format of the SSH Packets	626

20.3	FURTHER READING	626
	Books	626
	RFCs	627
20.4	KEY TERMS	627
20.5	SUMMARY	627
20.6	PRACTICE SET	628
	Exercises	628
	Research Activities	629

Chapter 21 File Transfer: FTP and TFTP 630

21.1	FTP	631
	Connections	631
	Communication	633
	Command Processing	635
	File Transfer	639
	Anonymous FTP	642
	Security for FTP	643
	The sftp Program	643
21.2	TFTP	643
	Messages	644
	Connection	646
	Data Transfer	647
	UDP Ports	649
	TFTP Example	650
	TFTP Options	650
	Security	651
	Applications	651
21.3	FURTHER READING	652
	Books	652
	RFCs	652
21.4	KEY TERMS	652
21.5	SUMMARY	653
21.6	PRACTICE SET	653
	Exercises	653
	Research Activities	655

Chapter 22 World Wide Web and HTTP 656

22.1	ARCHITECTURE	657
	Hypertext and Hypermedia	658
	Web Client (Browser)	658
	Web Server	659
	Uniform Resource Locator (URL)	659
22.2	WEB DOCUMENTS	660
	Static Documents	660
	Dynamic Documents	660
	Active Documents	663
22.3	HTTP	664
	HTTP Transaction	664

	Conditional Request	670
	Persistence	670
	Cookies	672
	Web Caching: Proxy Server	675
	HTTP Security	675
22.4	FURTHER READING	676
	Books	676
	RFCs	676
22.5	KEY TERMS	676
22.6	SUMMARY	676
22.7	PRACTICE SET	677
	Exercises	677
	Research Activities	678
Chapter 23 Electronic Mail: SMTP, POP, IMAP, and MIME 680		
23.1	ARCHITECTURE	681
	First Scenario	681
	Second Scenario	682
	Third Scenario	682
	Fourth Scenario	683
23.2	USER AGENT	684
	Services Provided by a User Agent	684
	User Agent Types	685
	Sending Mail	685
	Receiving Mail	686
	Addresses	686
	Mailing List or Group List	686
23.3	MESSAGE TRANSFER AGENT: SMTP	687
	Commands and Responses	687
	Mail Transfer Phases	691
23.4	MESSAGE ACCESS AGENT: POP AND IMAP	693
	POP3	694
	IMAP4	695
23.5	MIME	695
	MIME Headers	695
23.6	WEB-BASED MAIL	700
	Case I	700
	Case II	701
23.7	E-MAIL SECURITY	701
23.8	FURTHER READING	702
	Books	702
	RFCs	702
23.9	KEY TERMS	702
23.10	SUMMARY	702
23.11	PRACTICE SET	703
	Exercises	703
	Research Activities	704

Chapter 24 Network Management: SNMP 706

- 24.1 CONCEPT 707
 - Managers and Agents 707
- 24.2 MANAGEMENT COMPONENTS 708
 - Role of SNMP 708
 - Role of SMI 708
 - Role of MIB 709
 - An Analogy 709
 - An Overview 710
- 24.3 SMI 711
 - Name 711
 - Type 712
 - Encoding Method 713
- 24.4 MIB 715
 - Accessing MIB Variables 716
 - Lexicographic Ordering 718
- 24.5 SNMP 719
 - PDUs 719
 - Format 721
 - Messages 722
- 24.6 UDP PORTS 724
- 24.7 SECURITY 725
- 24.8 FURTHER READING 725
 - Books 725
 - RFCs 725
- 24.9 KEY TERMS 726
- 24.10 SUMMARY 726
- 24.11 PRACTICE SET 726
 - Exercises 726
 - Research Activity 727

Chapter 25 Multimedia 728

- 25.1 INTRODUCTION 729
- 25.2 DIGITIZING AUDIO AND VIDEO 730
 - Digitizing Audio 730
 - Digitizing Video 730
- 25.3 AUDIO AND VIDEO COMPRESSION 731
 - Audio Compression 731
 - Video Compression 731
- 25.4 STREAMING STORED AUDIO/VIDEO 736
 - First Approach: Using a Web Server 736
 - Second Approach: Using a Web Server with Metafile 737
 - Third Approach: Using a Media Server 738
 - Fourth Approach: Using a Media Server and RTSP 738
- 25.5 STREAMING LIVE AUDIO/VIDEO 739
- 25.6 REAL-TIME INTERACTIVE AUDIO/VIDEO 740
 - Characteristics 740

25.7	RTP	744
	RTP Packet Format	745
	UDP Port	746
25.8	RTCP	746
	Sender Report	746
	Receiver Report	747
	Source Description Message	747
	Bye Message	747
	Application-Specific Message	747
	UDP Port	747
25.9	VOICE OVER IP	748
	SIP	748
	H.323	750
25.10	QUALITY OF SERVICE	752
	Flow Characteristics	752
	Flow Classes	753
	Techniques to Improve QoS	753
	Resource Reservation	757
	Admission Control	758
25.11	INTEGRATED SERVICES	758
	Signaling	758
	Flow Specification	758
	Admission	759
	Service Classes	759
	RSVP	759
	Problems with Integrated Services	762
25.12	DIFFERENTIATED SERVICES	762
	DS Field	762
25.13	RECOMMENDED READING	764
	Books	764
	RFCs	764
25.14	KEY TERMS	764
25.15	SUMMARY	765
25.16	PRACTICE SET	766
	Exercises	766

Part 5 Next Generation 767

Chapter 26 IPv6 Addressing 768

26.1	INTRODUCTION	769
	Notations	769
	Address Space	772
	Three Address Types	772
	Broadcasting and Multicasting	773
26.2	ADDRESS SPACE ALLOCATION	773
	Assigned and Reserved Blocks	775
26.3	GLOBAL UNICAST ADDRESSES	778
	Three Levels of Hierarchy	779

26.4	AUTOCONFIGURATION	781
26.5	RENUMBERING	782
26.6	FURTHER READING	782
	Books	782
	RFCs	782
26.7	KEY TERMS	783
26.8	SUMMARY	783
26.9	PRACTICE SET	783
	Exercises	783

Chapter 27 IPv6 Protocol 786

27.1	INTRODUCTION	787
	Rationale for Change	787
	Reason for Delay in Adoption	787
27.2	PACKET FORMAT	788
	Base Header	788
	Flow Label	789
	Comparison between IPv4 and IPv6 Headers	790
	Extension Headers	790
	Comparison between IPv4 and IPv6	795
27.3	TRANSITION FROM IPv4 TO IPv6	796
	Dual Stack	796
	Tunneling	797
	Header Translation	797
27.4	FURTHER READING	798
	Books	798
	RFCs	798
27.5	KEY TERMS	798
27.6	SUMMARY	799
27.7	PRACTICE SET	799
	Exercises	799
	Research Activity	799

Chapter 28 ICMPv6 800

28.1	INTRODUCTION	801
28.2	ERROR MESSAGES	802
	Destination-Unreachable Message	802
	Packet-Too-Big Message	803
	Time-Exceeded Message	803
	Parameter-Problem Message	804
28.3	INFORMATIONAL MESSAGES	804
	Echo-Request Message	804
	Echo-Reply Message	805
28.4	NEIGHBOR-DISCOVERY MESSAGES	805
	Router-Solicitation Message	805
	Router-Advertisement Message	806
	Neighbor-Solicitation Message	806

	Neighbor-Advertisement Message	807
	Redirection Message	808
	Inverse-Neighbor-Solicitation Message	808
	Inverse-Neighbor-Advertisement Message	808
28.5	GROUP MEMBERSHIP MESSAGES	809
	Membership-Query Message	809
	Membership-Report Message	810
	Functionality	810
28.6	FURTHER READING	812
	Books	812
	RFCs	812
28.7	KEY TERMS	812
28.8	SUMMARY	812
28.9	PRACTICE SET	813
	Exercises	813
	Research Activities	813

Part 6 Security 815

Chapter 29 Cryptography and Network Security 816

29.1	INTRODUCTION	817
	Security Goals	817
	Attacks	818
	Services	819
	Techniques	819
29.2	TRADITIONAL CIPHERS	820
	Key	821
	Substitution Ciphers	821
	Transposition Ciphers	824
	Stream and Block Ciphers	825
29.3	MODERN CIPHERS	826
	Modern Block Ciphers	826
	Data Encryption Standard (DES)	828
	Modern Stream Ciphers	830
29.4	ASYMMETRIC-KEY CIPHERS	831
	Keys	832
	General Idea	832
	RSA Cryptosystem	834
	Applications	836
29.5	MESSAGE INTEGRITY	836
	Message and Message Digest	836
	Hash Functions	837
29.6	MESSAGE AUTHENTICATION	838
	HMAC	838
29.7	DIGITAL SIGNATURE	839
	Comparison	839
	Process	840

	Signing the Digest	841
	Services	842
	RSA Digital Signature Scheme	843
	Digital Signature Standard (DSS)	844
29.8	ENTITY AUTHENTICATION	844
	Entity versus Message Authentication	844
	Verification Categories	845
	Passwords	845
	Challenge-Response	845
29.9	KEY MANAGEMENT	847
	Symmetric-Key Distribution	847
	Symmetric-Key Agreement	850
	Public-Key Distribution	851
29.10	FURTHER READING	853
29.11	KEY TERMS	853
29.12	SUMMARY	854
29.13	PRACTICE SET	855
	Exercises	855
	Research Activities	856

Chapter 30 Internet Security 858

30.1	NETWORK LAYER SECURITY	859
	Two Modes	859
	Two Security Protocols	861
	Services Provided by IPSec	864
	Security Association	865
	Internet Key Exchange (IKE)	868
	Virtual Private Network (VPN)	868
30.2	TRANSPORT LAYER SECURITY	869
	SSL Architecture	869
	Four Protocols	872
30.3	APPLICATION LAYER SECURITY	875
	E-mail Security	875
	Pretty Good Privacy (PGP)	876
	Key Rings	878
	PGP Certificates	878
	S/MIME	881
	Applications of S/MIME	885
30.4	FIREWALLS	885
	Packet-Filter Firewall	885
	Proxy Firewall	886
30.5	RECOMMENDED READING	887
30.6	KEY TERMS	887
30.7	SUMMARY	888
30.8	PRACTICE SET	888
	Exercises	888
	Research Activities	889

Part 7 Appendices 891

Appendix A Unicode 892

Appendix B Positional Numbering Systems 896

Appendix C Error Detection Codes 904

Appendix D Checksum 914

Appendix E HTML, XHTML, XML, and XSL 920

Appendix F Client-Server Programming in Java 926

Appendix G Miscellaneous Information 932

Glossary 935

References 955

Index 957

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 Valero, *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.

P A R T

1

Introduction and Underlying Technologies

Chapter 1 Introduction 2

Chapter 2 The OSI Model and the TCP/IP Protocol Suite 18

Chapter 3 Underlying Technologies 46

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.

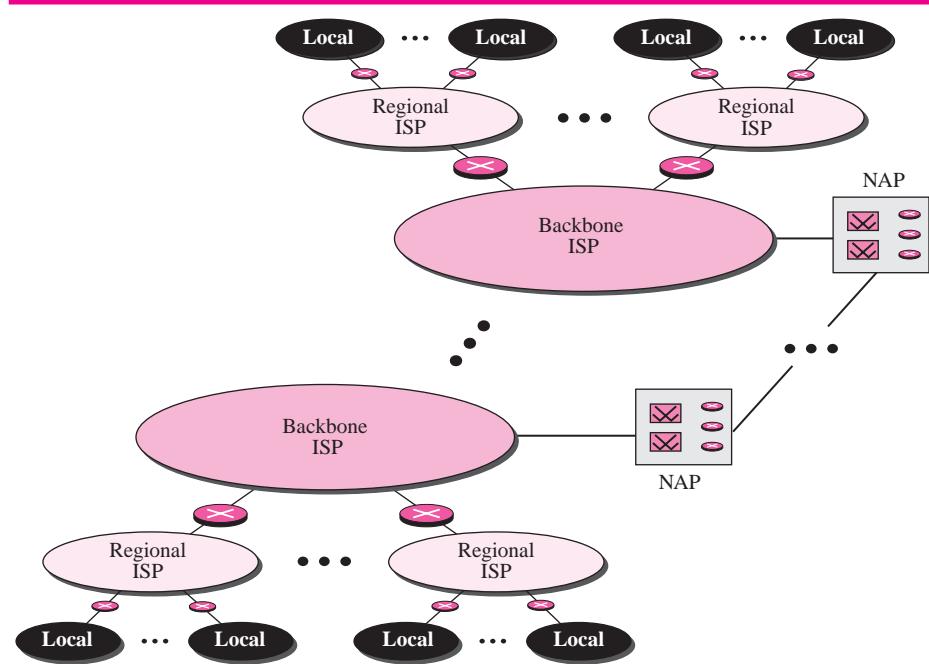
ANSNET

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 Sprint-Link, 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 organization 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

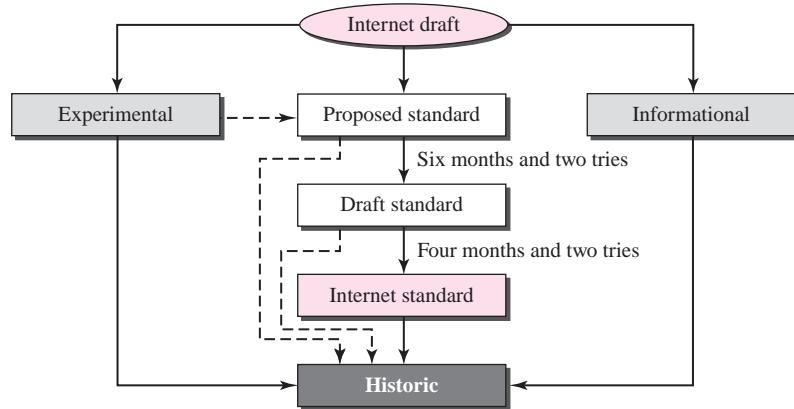
Internet authorities, a draft may be published as a **Request for Comment (RFC)**. Each RFC is edited, assigned a number, and made available to all interested parties.

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.

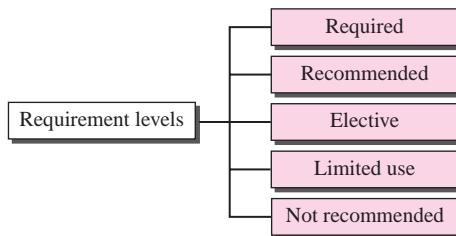
Informational

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.

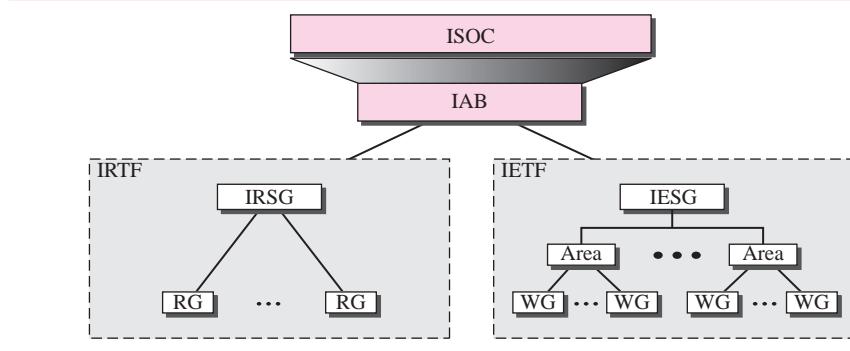
Not Recommended

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

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)	Internet draft
American National Standards Institute (ANSI)	Internet Engineering Task Force (IETF)
ANSNET	Internet Research Task Force (IRTF)
ARPANET	Internet Service Provider (ISP)
ATM Forum	Internet Society (ISOC)
Consultative Committee for International Telegraphy and Telephony (CCITT)	Internet standard
CSNET	Internet Protocol (IP)
de facto standards	maturity levels
de jure standards	MILNET
Electronic Industries Association (EIA)	network
Federal Communications Commission (FCC)	network access points (NAPs)
Frame Relay Forum	Network Information Center (NIC)
Institute of Electrical and Electronics Engineers (IEEE)	NSFNET
International Standards Organization (ISO)	protocol
International Telecommunications Union–Telecommunications Standards Sector (ITU-T)	Open Mobile Alliance (OMA)
Internet Architecture Board (IAB)	Request for Comment (RFC)
Internet Assigned Numbers Authority (IANA)	requirement levels
Internet Corporation for Assigned Names and Numbers (ICANN)	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.

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.

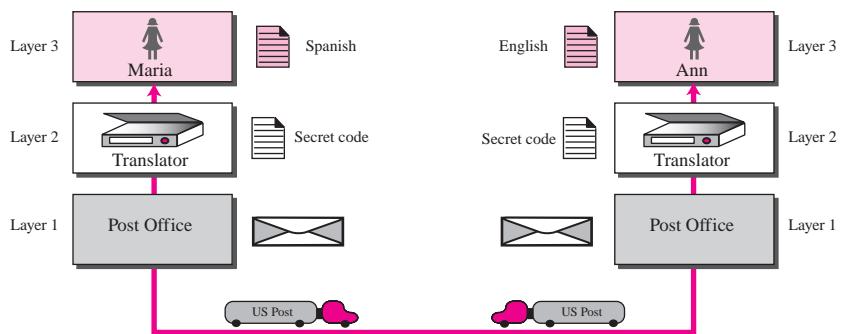
Figure 2.1 Example 2.1



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 multi-national 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.

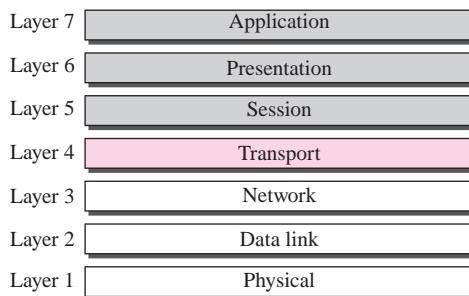
ISO is the organization; OSI is the model.

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*

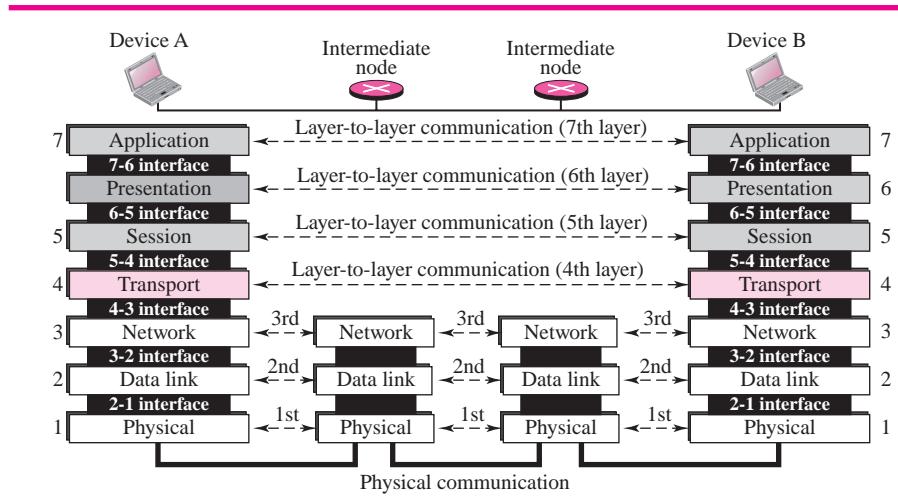


Layered Architecture

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.

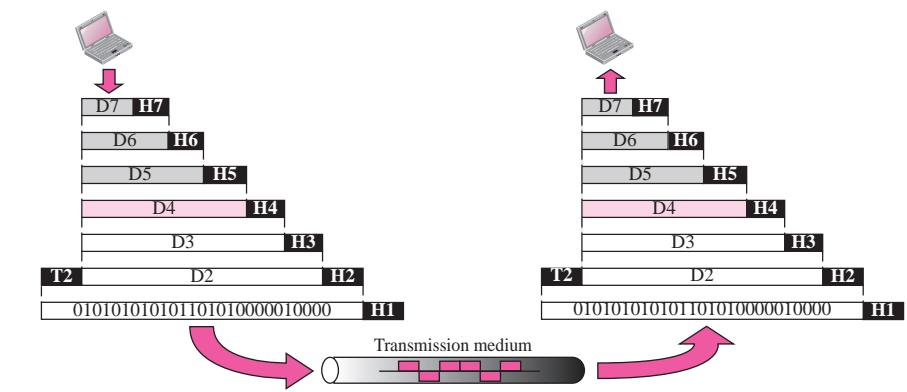
Organization of the Layers

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

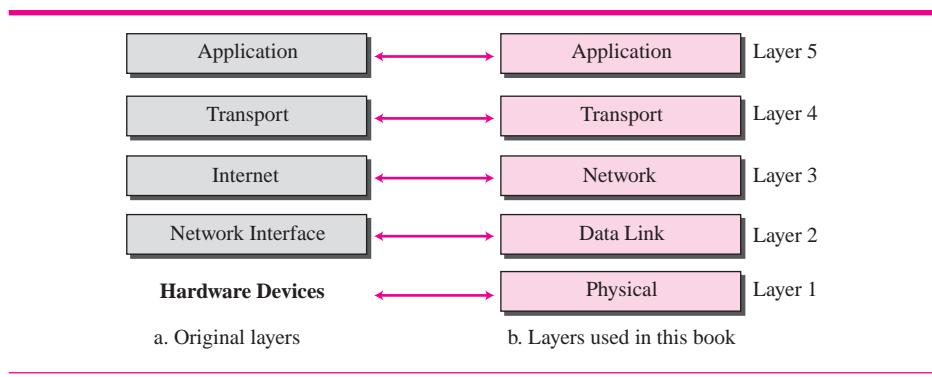
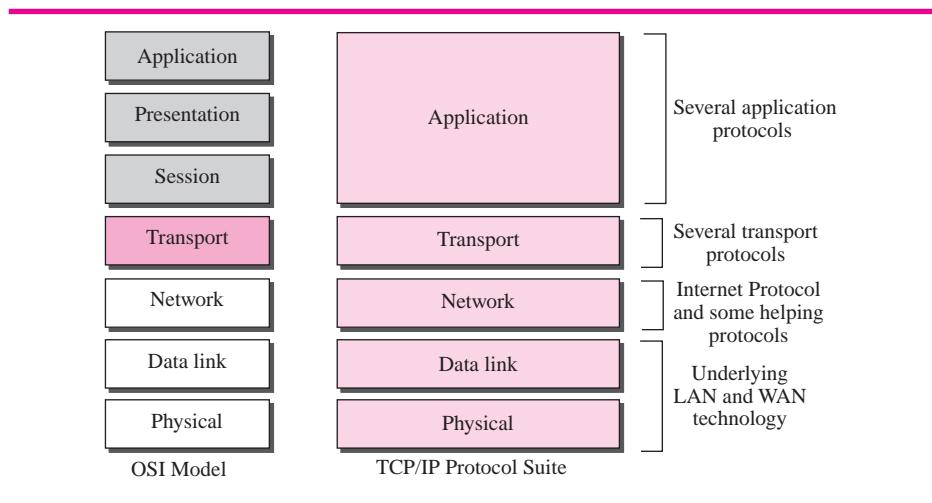
Application	To allow access to network resources	7
Presentation	To translate, encrypt, and compress data	6
Session	To establish, manage, and terminate sessions	5
Transport	To provide reliable process-to-process message delivery and error recovery	4
Network	To move packets from source to destination; to provide internetworking	3
Data link	To organize bits into frames; to provide hop-to-hop delivery	2
Physical	To transmit bits over a medium; to provide mechanical and electrical specifications	1

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

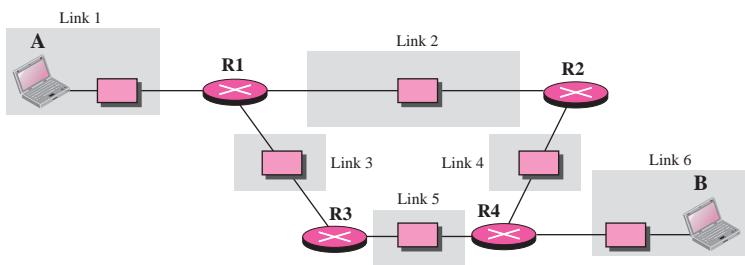
Two reasons were mentioned for this decision. First, TCP/IP has more than one transport-layer protocol. Some of the functionalities of the session layer are available in some of the transport layer protocols. Second, the application layer is not only one piece of software. Many applications can be developed 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.

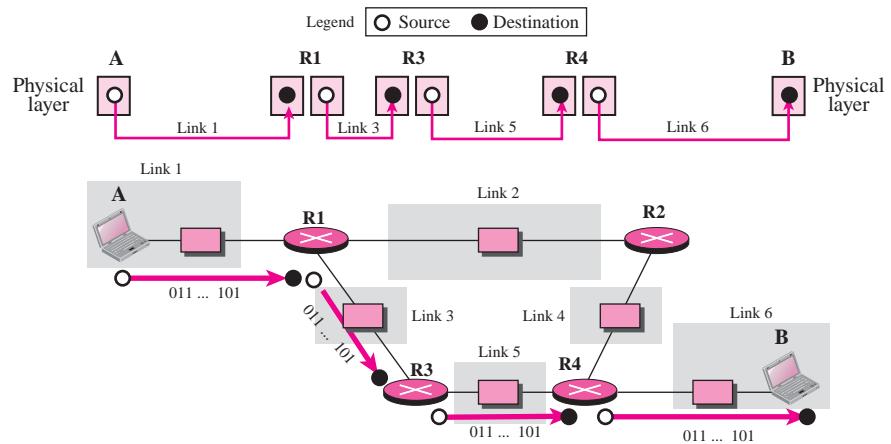
Figure 2.9 A private internet



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.

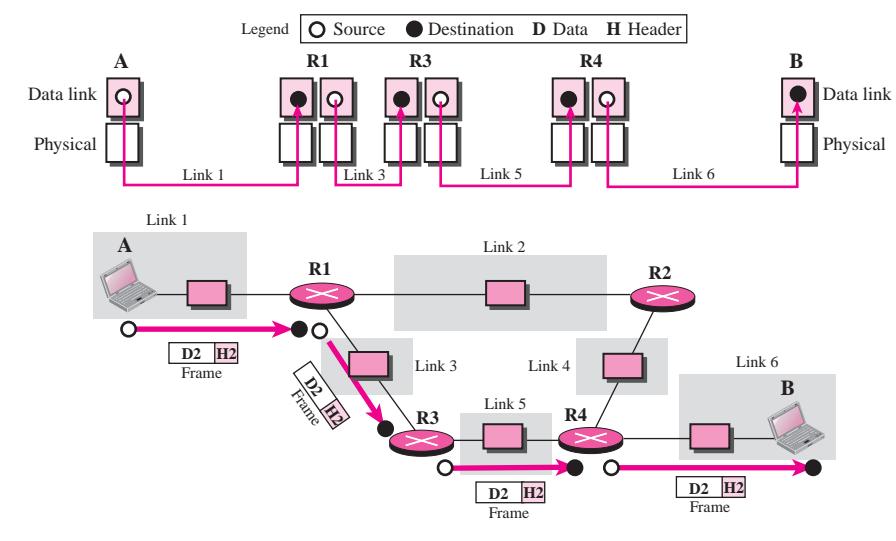
The unit of communication at the physical layer is a bit.

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

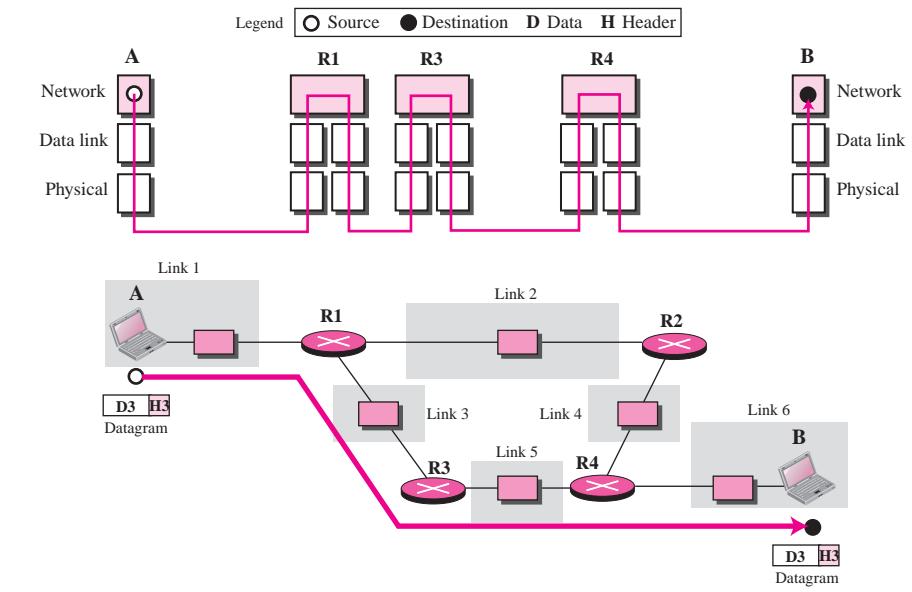
link layer protocol shown at the right to create a new frame to be sent to the router R3. The reason is that the two links, link 1 and link 3, may be using different protocols and require frames of different formats. Note also that the figure does not show the physical movement of frames; the physical 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.

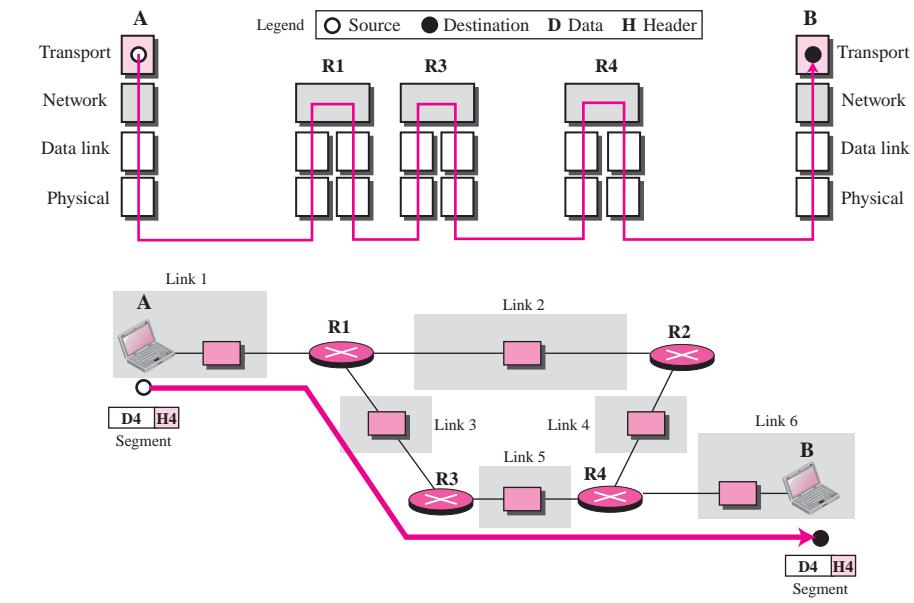
The unit of communication at the network layer is a datagram.

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



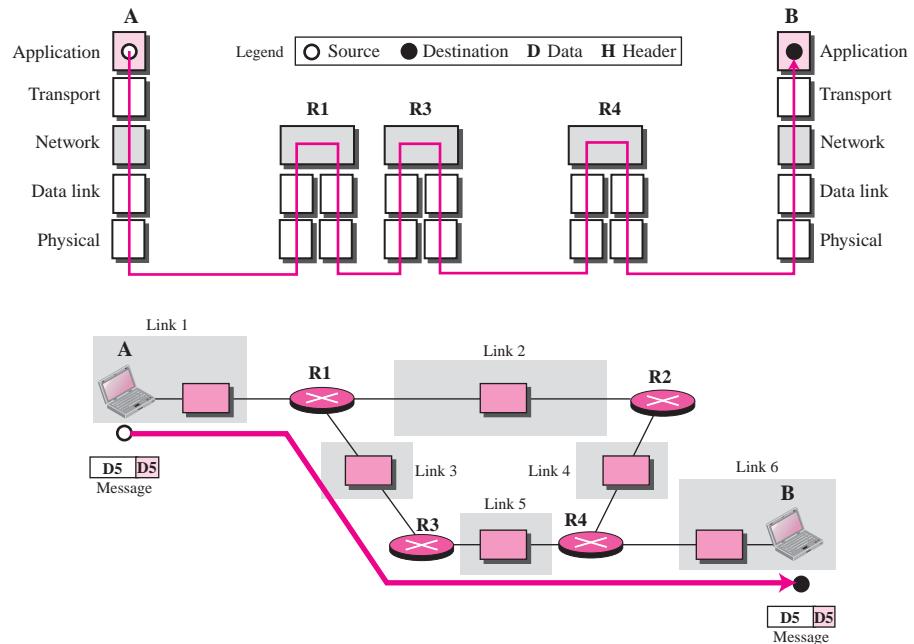
Again, we should know that the two transport layers only think that they are communicating with each other using a segment; the communication is done through the physical layer and the exchange of bits.

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.

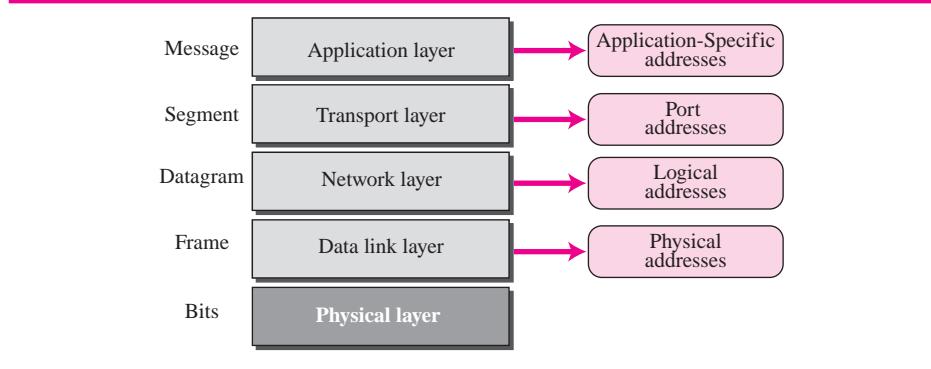
The unit of communication at the application layer is a message.

2.4 ADDRESSING

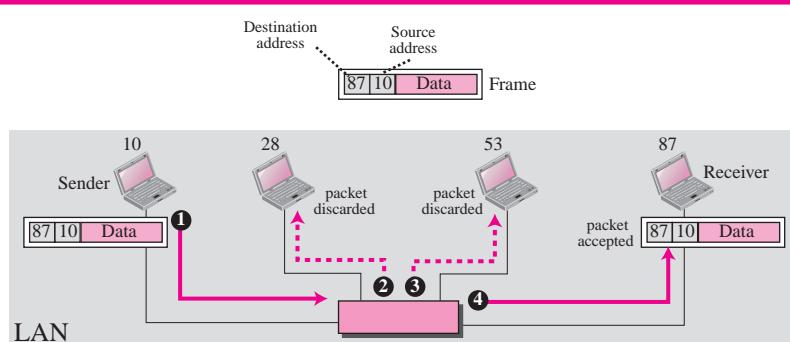
Four levels of addresses are used in an internet employing the TCP/IP protocols: **physical address**, **logical address**, **port address**, and **application-specific address**. Each address is related to a one layer in the TCP/IP architecture, as shown in Figure 2.15.

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.

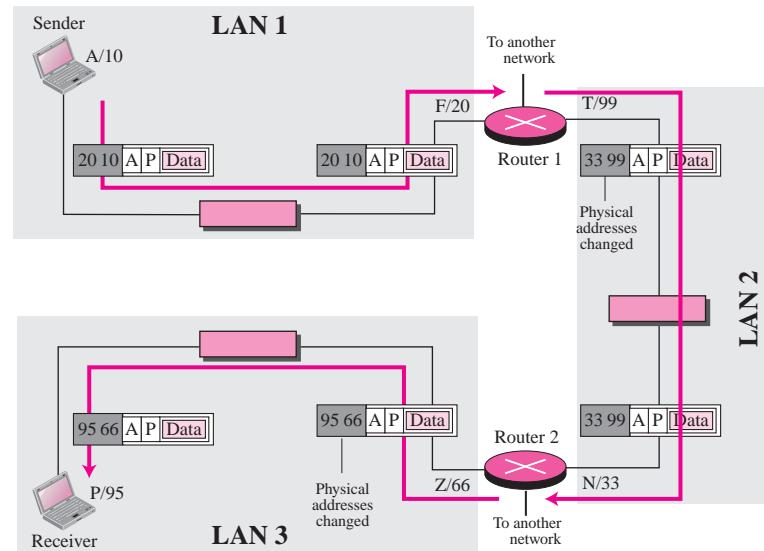
Example 2.5

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.

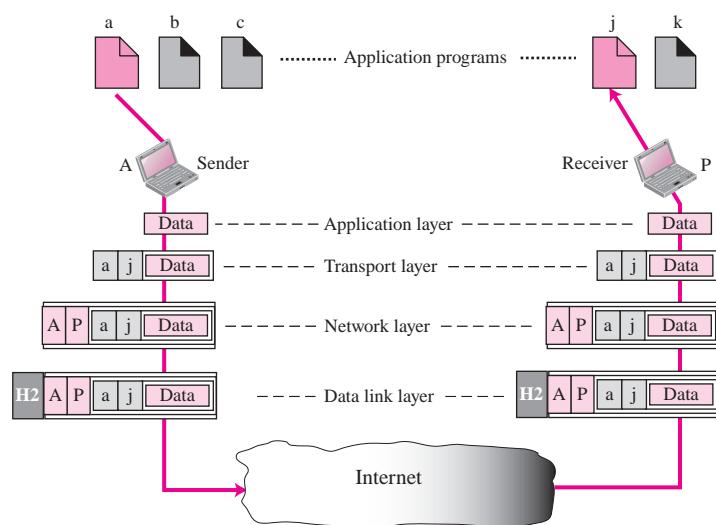
Port Addresses

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

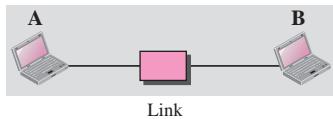
2.8 PRACTICE SET

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

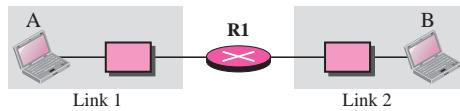
- c. ensures reliable transmission of data
 - d. log-in and log-out procedures
 - e. provides independence from differences in data representation
6. Show the communication at the application layer (see Figure 2.14) for the simple private internet in Figure 2.19.

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.

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.

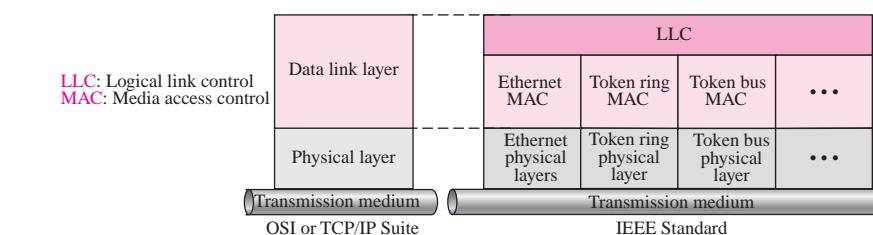
IEEE Standards

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.

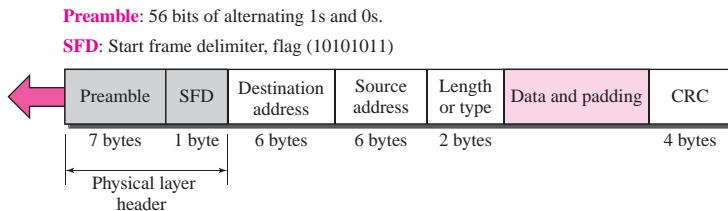
Frame Format

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



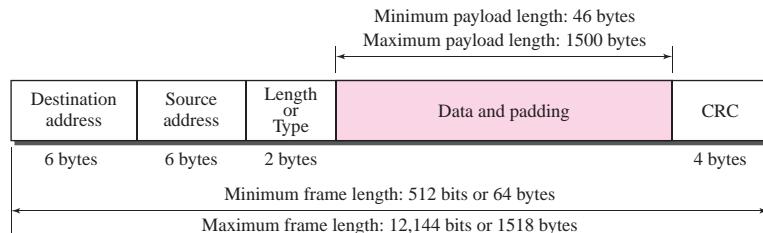
- ❑ **Preamble.** The first field of the 802.3 frame contains 7 bytes (56 bits) of alternating 0s and 1s that alerts the receiving system to the coming frame and enables it to synchronize its input timing. The pattern provides only an alert and a timing pulse. The 56-bit pattern allows the stations to 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.

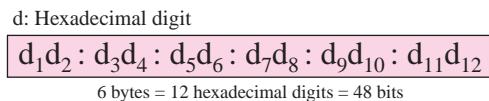
The standard defines the maximum length of a frame (without preamble and SFD field) as 1518 bytes. If we subtract the 18 bytes of header and trailer, the maximum length of the payload is 1500 bytes. The maximum length restriction has two historical reasons. First, memory was very expensive when Ethernet 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.

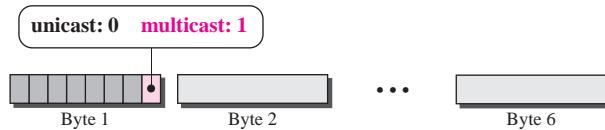
Figure 3.4 Ethernet address in hexadecimal notation

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:

- 4A:30:10:21:10:1A
- 47:20:1B:2E:08:EE
- 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:

- This is a unicast address because A in binary is 1010 (even).
- This is a multicast address because 7 in binary is 0111 (odd).
- 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

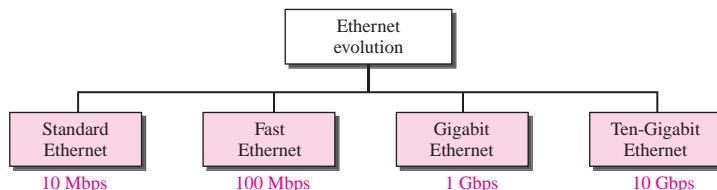
The address is sent left-to-right, byte by byte; for each byte, it is sent right-to-left, bit by bit, as shown below:

```
← 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

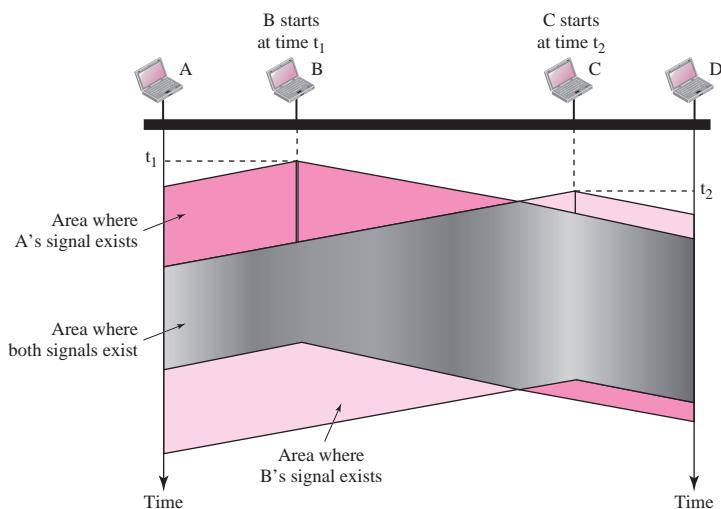
The original Ethernet with 10-Mbps data rate is now history, but we briefly discuss its characteristics to pave the way for understanding other Ethernet versions.

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

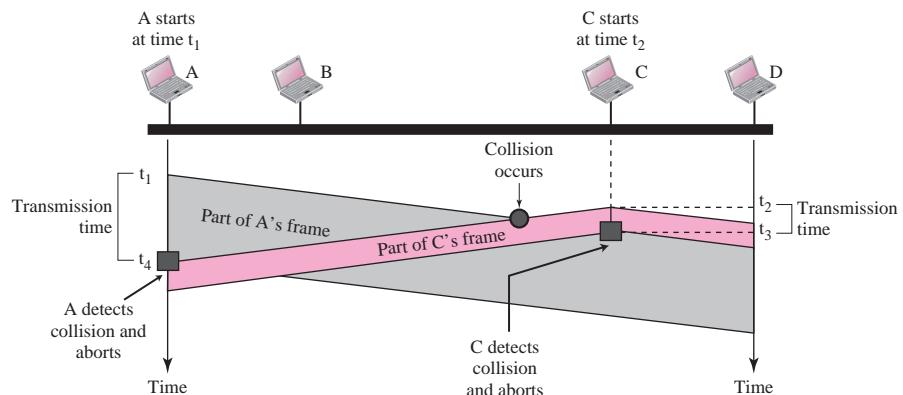


The possibility of collision still exists because of propagation delay; when a station sends a frame, it still takes time (although very short) for the first bit to reach every station and for every station to sense it. In other words, a station may sense the medium and find it idle, only because the 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 $25.6 \mu\text{s}$, what is the minimum size of the frame?

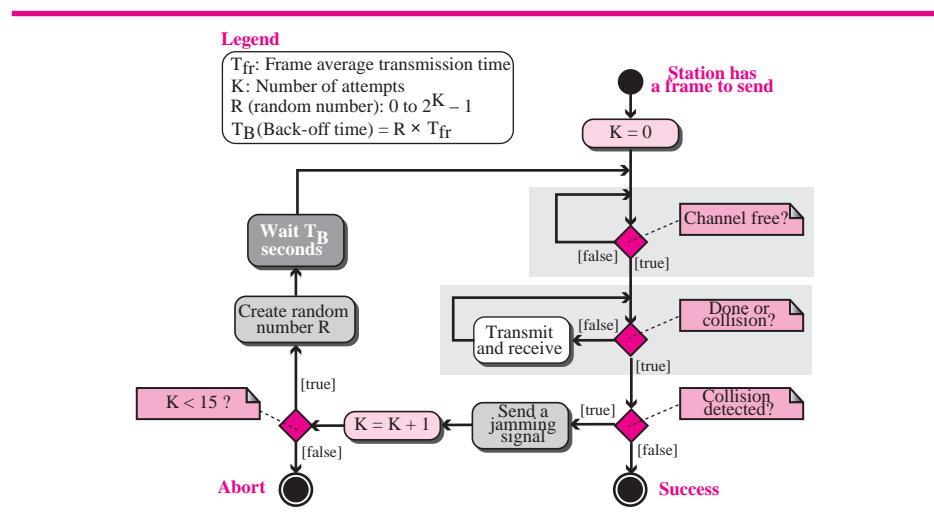
Solution

The frame transmission time is $T_{fr} = 2 \times T_p = 51.2 \mu\text{s}$. This means, in the worst case, a station needs to transmit for a period of $51.2 \mu\text{s}$ to detect the collision. The minimum size of the frame is $10 \text{ Mbps} \times 51.2 \mu\text{s} = 512 \text{ bits}$ 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

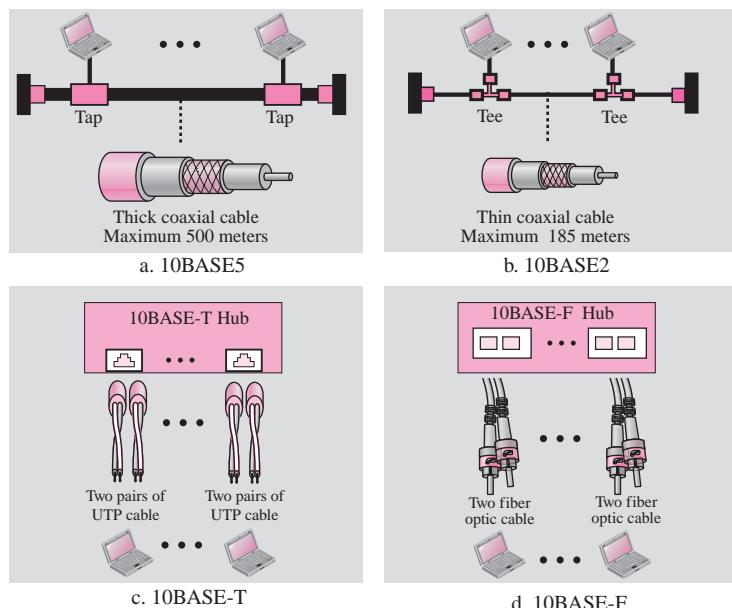
The Standard Ethernet defined several implementations, but only four of them became popular during '80s. Table 3.1 shows a summary of Standard Ethernet implementations. In the nomenclature 10Base-X, the number defines the data rate (10 Mbps), the term Base means baseband (digital) signal, and X approximately 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

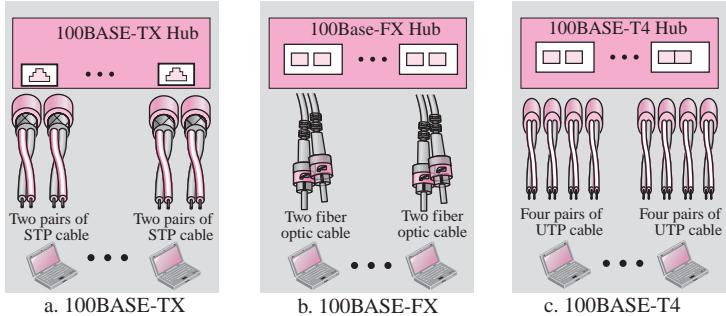
Characteristics	100Base-TX	100Base-FX	100Base-T4
Media	STP	Fiber	UTP
Number of wires	2	2	4
Maximum length	100 m	100 m	100 m

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

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

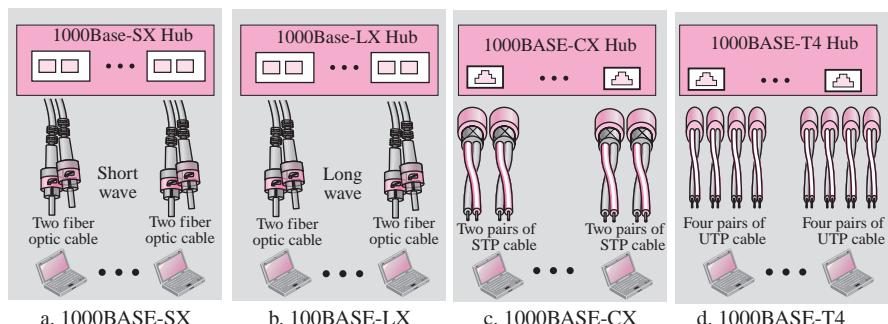
Table 3.3 is a summary of the Gigabit Ethernet implementations.

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



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.

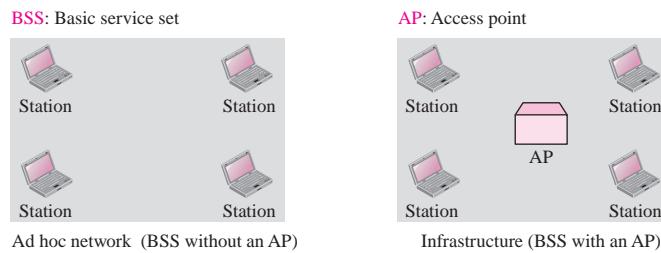
Architecture

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

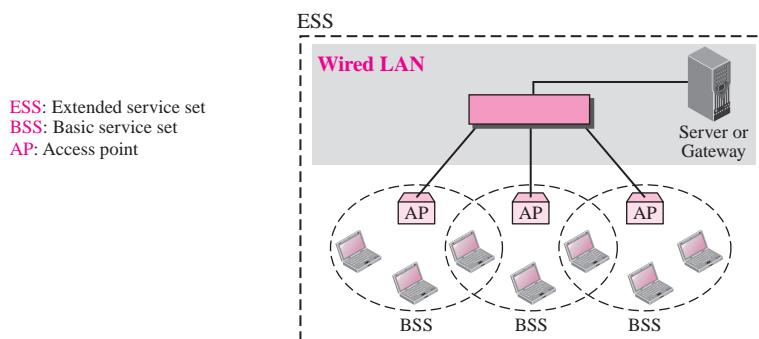
cannot send data to other BSSs. It is called an *ad hoc architecture*. In this architecture, stations can form a network without the need of an AP; they can locate one another and agree to be part of a BSS. A BSS with an AP is sometimes referred to as an *infrastructure* network.

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

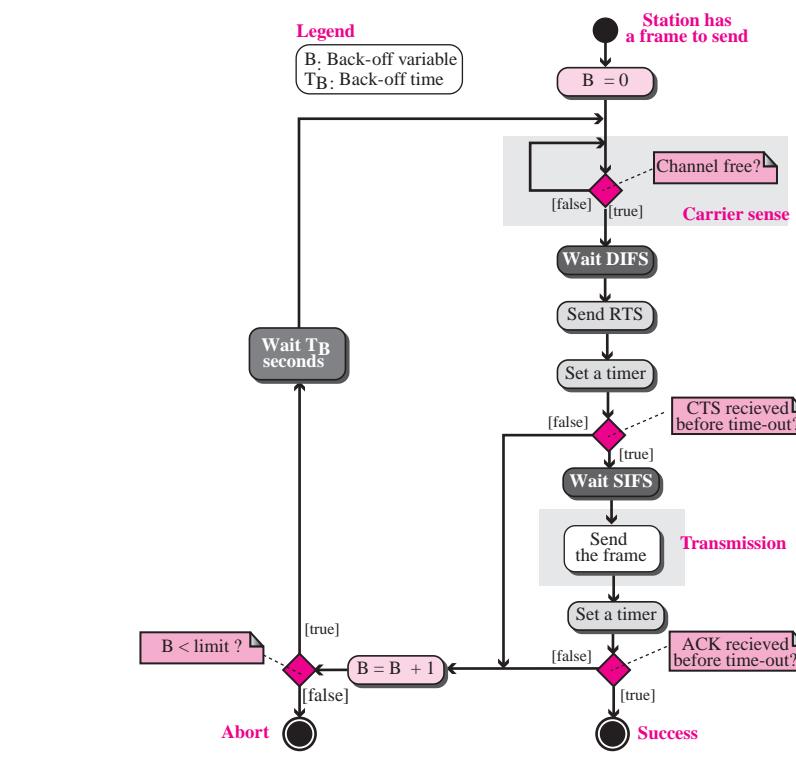
IEEE 802.11 defines three types of stations based on their mobility in a wireless LAN: **no-transition**, **BSS-transition**, and **ESS-transition mobility**. A station with no-transition

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



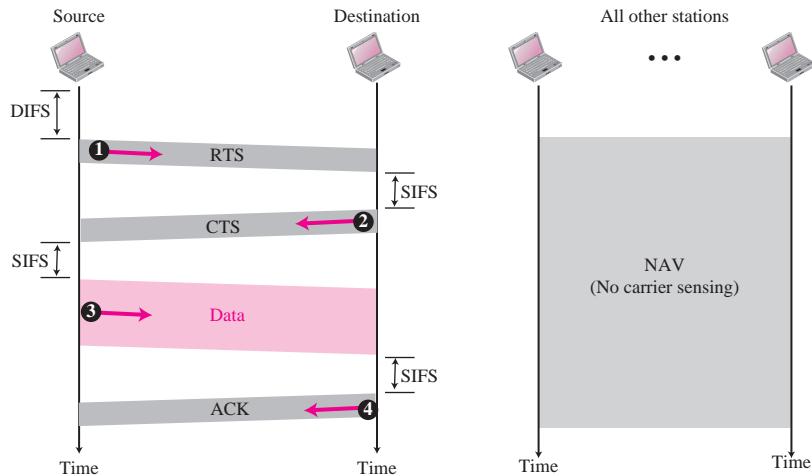
Wireless LANs cannot implement CSMA/CD for three reasons:

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.

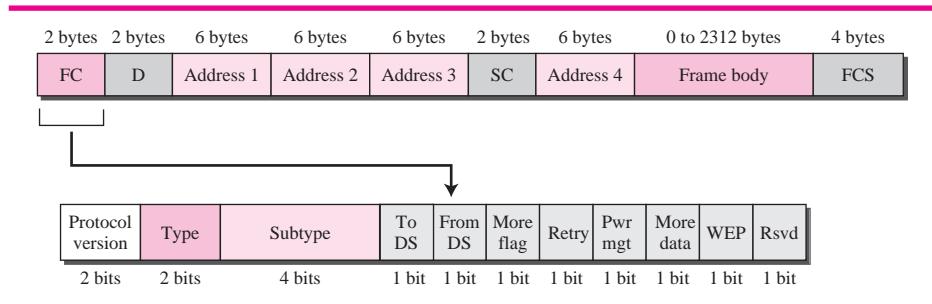
Fragmentation

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.

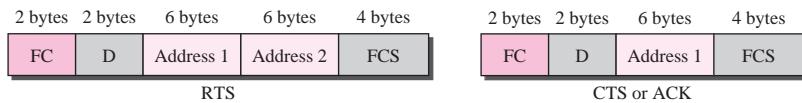
Frame Types

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

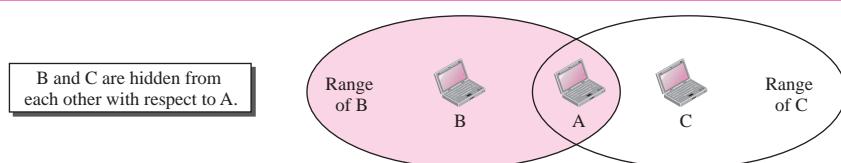
To DS	From DS	Address 1	Address 2	Address 3	Address 4
0	0	Destination	Source	BSS ID	N/A
0	1	Destination	Sending AP	Source	N/A
1	0	Receiving AP	Source	Destination	N/A
1	1	Receiving AP	Sending AP	Destination	Source

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

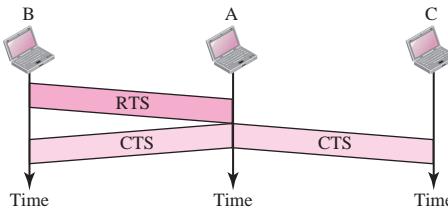
Assume that station B is sending data to station A. In the middle of this transmission, station C also has data to send to station A. However, station C is out of B's range and transmissions from B cannot reach C. Therefore C thinks the medium is free. Station C sends its data to A, which results in 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.

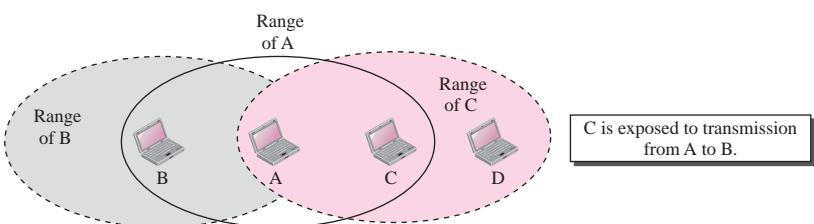
The CTS frame in CSMA/CA handshake can prevent collision from a hidden station.

Figure 3.20 Use of handshaking to prevent hidden station problem



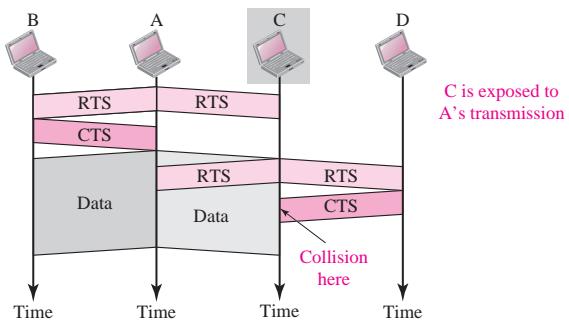
Exposed Station Problem Now consider a situation that is the inverse of the previous one: the exposed station problem. In this problem a station refrains from using a channel when it is, in fact, available. In Figure 3.21, station A is transmitting to station B. Station C has some data to send to station 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.

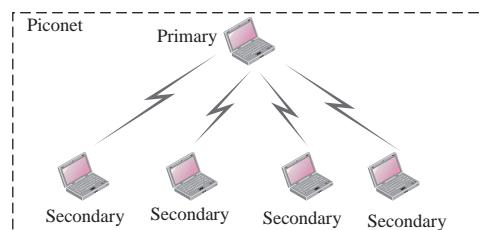
Today, Bluetooth technology is the implementation of a protocol defined by the IEEE 802.15 standard. The standard defines a wireless personal area network (PAN) operable in an area the size of a room or a hall.

Architecture

Bluetooth defines two types of networks: piconet and scatternet.

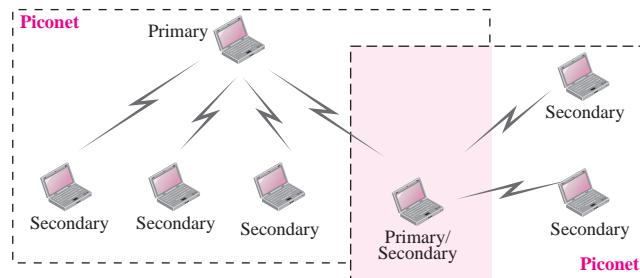
Piconets A Bluetooth network is called a **piconet**, or a small net. A piconet can have up to eight stations, one of which is called the **primary**; the rest are called **secondaries**. All the secondary stations synchronize their clocks and hopping sequence with the primary. Note that a piconet can 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

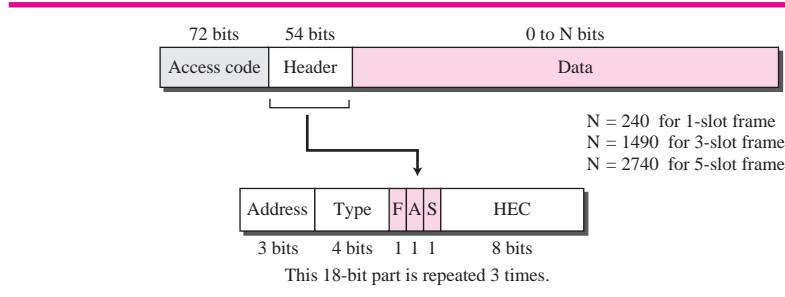
A Bluetooth device has a built-in short-range radio transmitter. The current data rate is 1 Mbps with a 2.4-GHz bandwidth. This means that there is a possibility of interference between the IEEE 802.11b wireless LANs and Bluetooth LANs.

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 μ s. However, in a one-slot frame exchange, 259 μ s is needed for hopping and control mechanisms. This means that a one-slot frame can last only 625 – 259, or 366 μ s. 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 μ s is used for hopping, the length of the frame is $3 \times 625 - 259 = 1616 \mu$ s 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.
- 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.
 - The receiver acknowledges the request (broadcast to all stations) by sending a small packet called clear to send (CTS).
 - The sender sends the data frame.
 - The receiver acknowledges the receipt of data.

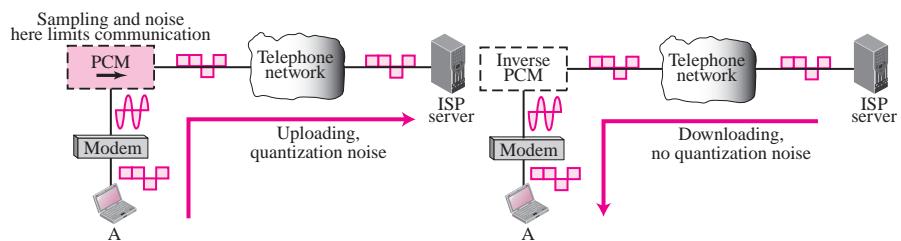
3.3 POINT-TO-POINT WANS

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 8000×7 , 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

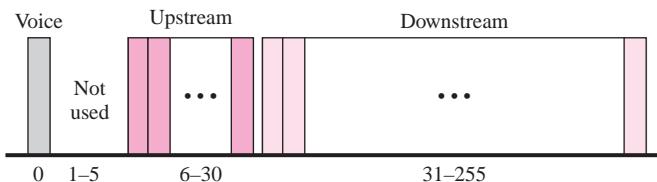
The first technology in the set is **asymmetric DSL (ADSL)**. ADSL, like a 56K modem, provides higher speed (bit rate) in the downstream direction (from the Internet to the resident) than in the upstream direction (from the resident to the Internet). That is the reason it is called asymmetric. Unlike 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 a 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



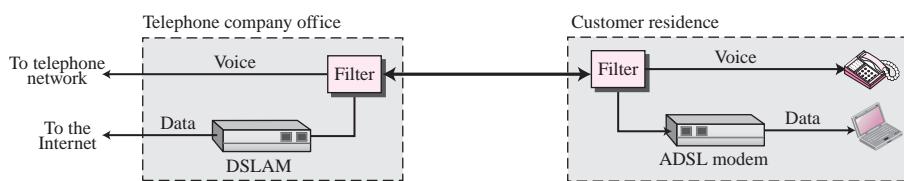
Because of the high signal/noise ratio, the actual bit rate is much lower than the above-mentioned rates. The bit rates are as follows:

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.

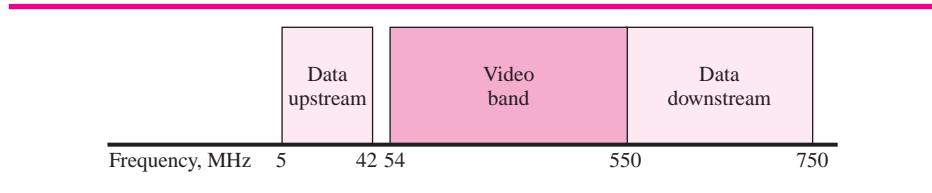
HFC Network

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 ($2 \text{ bits/Hz} \times 6 \text{ MHz}$). However, the data rate is usually less than 12 Mbps.

Sharing

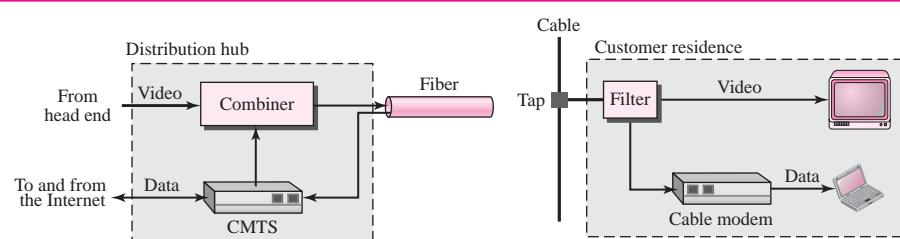
Both upstream and downstream bands are shared by the subscribers. The upstream data bandwidth is only 37 MHz. This means that there are only six 6-MHz channels available in the upstream direction. A subscriber needs to use one channel to send data in the upstream direction. The question is, How can six 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

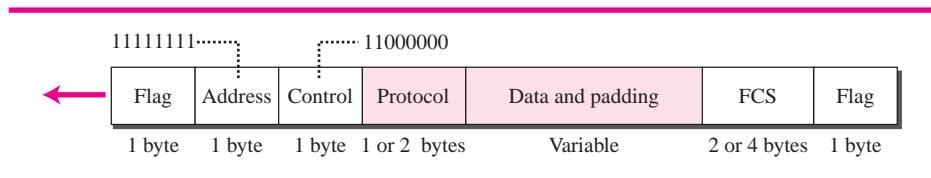
PPP

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:

- Flag field.** The flag field identifies the boundaries of a PPP frame. Its value is 01111110.
- 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.
- 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.
- Protocol field.** The protocol field defines the type of data being carried in the data field: user data or other information.
- Data field.** This field carries either user data or other information.
- 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.

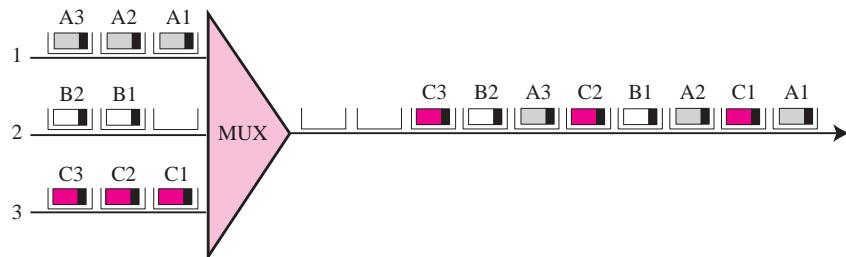
A cell network uses the cell as the basic unit of data exchange.
A cell is defined as a small, fixed-size block of information.

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.

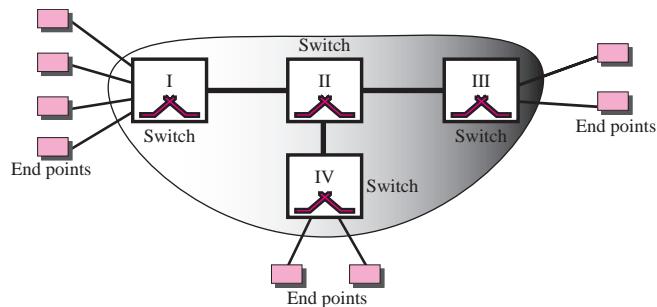
Figure 3.32 ATM multiplexing



ATM Architecture

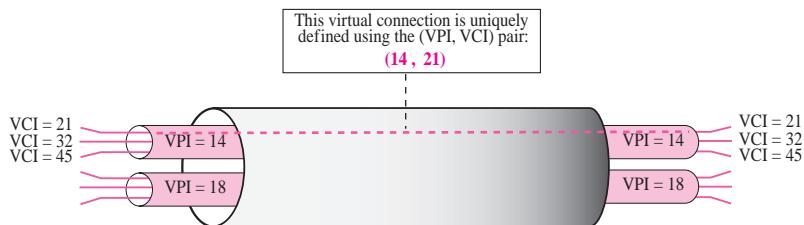
ATM is a switched network. The user access devices, called the end points, are connected to the switches inside the network. The switches are connected to each other using high-speed communication channels. Figure 3.33 shows an example of an ATM network.

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.

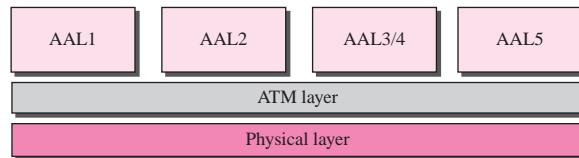
In a virtual circuit network, to route data from one end point to another, the virtual connections need to be identified. For this purpose, the designers of ATM created a hierarchical identifier with two levels: a **virtual path identifier (VPI)** and a **virtual circuit identifier (VCI)**. The VPI defines 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

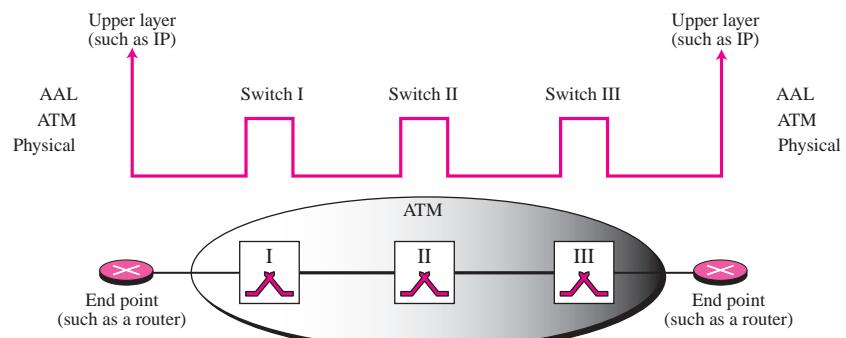
The ATM standard defines three layers. They are, from top to bottom, the application adaptation layer, the ATM layer, and the physical layer as shown in Figure 3.35.

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.

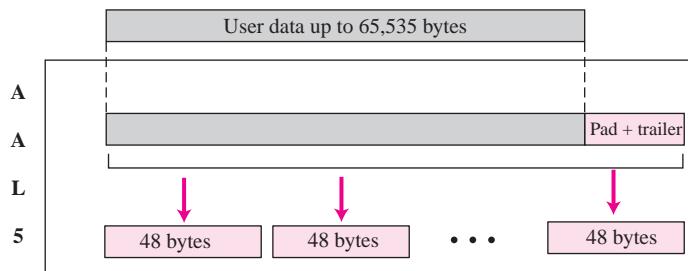
AAL5, which is sometimes called the **simple and efficient adaptation layer (SEAL)**, assumes that all cells belonging to a single message travel sequentially and that control functions are included in the upper layers of the sending application. AAL5

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.

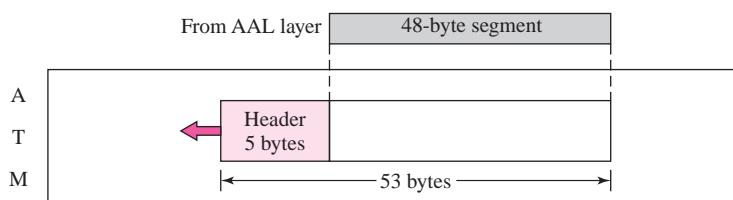
Figure 3.37 AAL5



ATM Layer

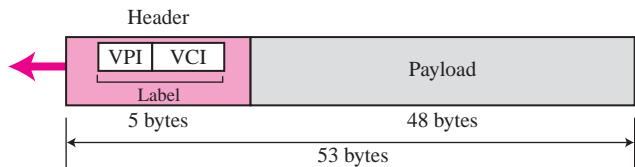
The ATM layer provides routing, traffic management, switching, and multiplexing services. It processes outgoing traffic by accepting 48-byte segments from the AAL sublayer. The addition of a 5-byte header transforms the segment into a 53-byte cell (see Figure 3.38).

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.

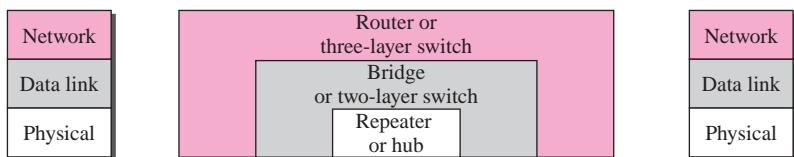
Figure 3.39 An ATM cell

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.

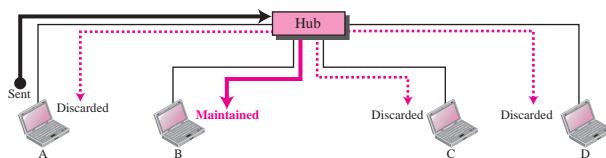
Figure 3.40 Connecting devices

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

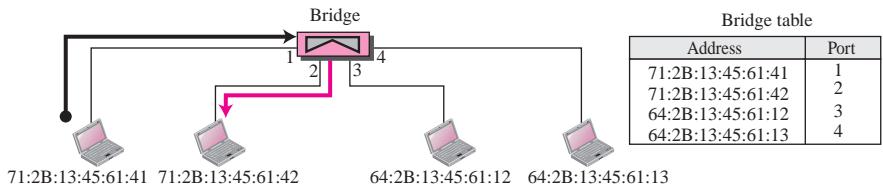
A **bridge** operates in both the physical and the data link layers. As a physical-layer device, it regenerates the signal it receives. As a data link layer device, the bridge can check the MAC addresses (source and destination) contained in the frame.

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 sent 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:

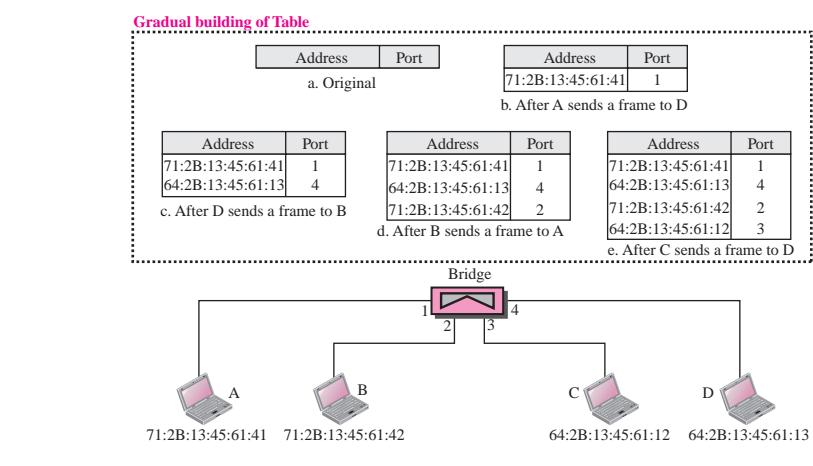
1. Frames must be forwarded from one station to another.
2. The forwarding table is automatically made by learning frame movements in the network.
3. Loops in the system must be prevented.

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.

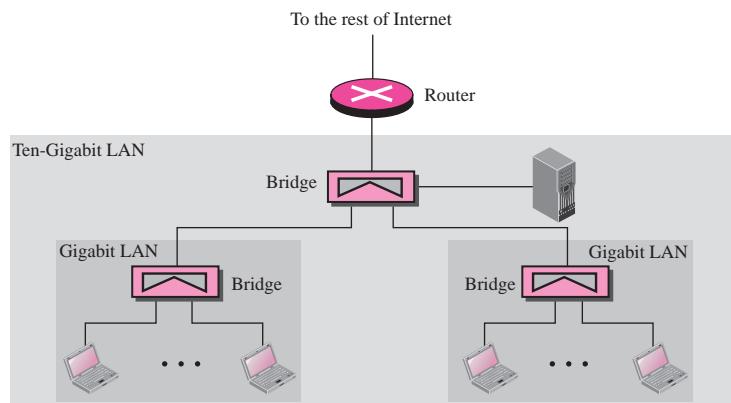
A repeater or a bridge connects segments of a LAN.
A router connects independent LANs or WANs to create an internetwork (internet).

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

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

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.

P A R T

2

Network Layer

Chapter 4	Introduction to Network Layer	94
Chapter 5	IPv4 Addresses	114
Chapter 6	Delivery and Forwarding of IP Packets	160
Chapter 7	Internet Protocol Version 4 (IPv4)	186
Chapter 8	Address Resolution Protocol (ARP)	220
Chapter 9	Internet Control Message Protocol Version 4 (ICMPv4)	244
Chapter 10	Mobile IP	268
Chapter 11	Unicast Routing Protocols (RIP, OSPF, and BGP)	282
Chapter 12	Multicasting and Multicast Routing Protocols	334

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

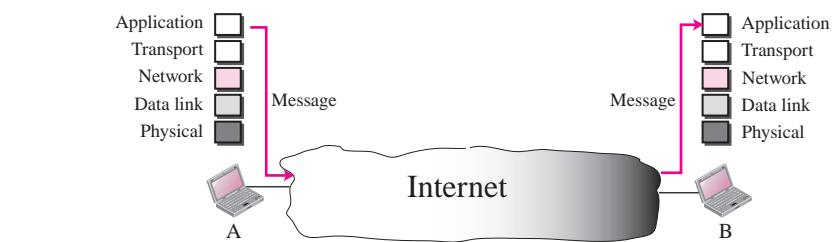
This chapter has several objectives:

- ❑ 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

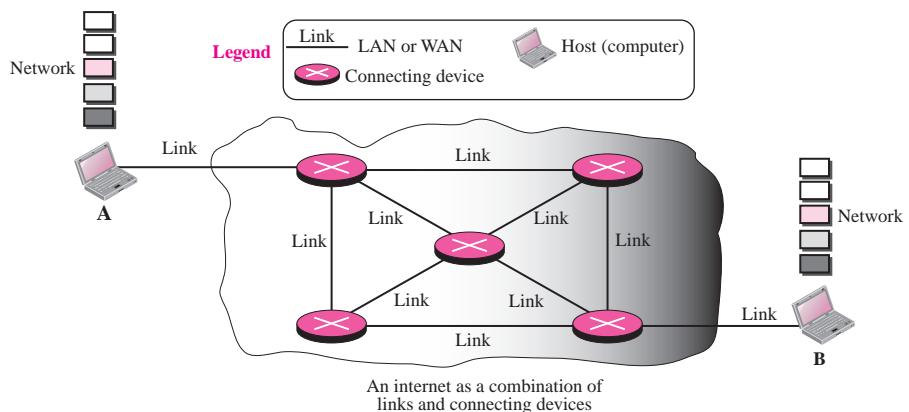
At the conceptual level, we can think of the global Internet as a black box network that connects millions (if not billions) of computers in the world together. At this level, we are only concerned that a message from the application layer in one computer reaches the application layer in another computer.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



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 sent 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.

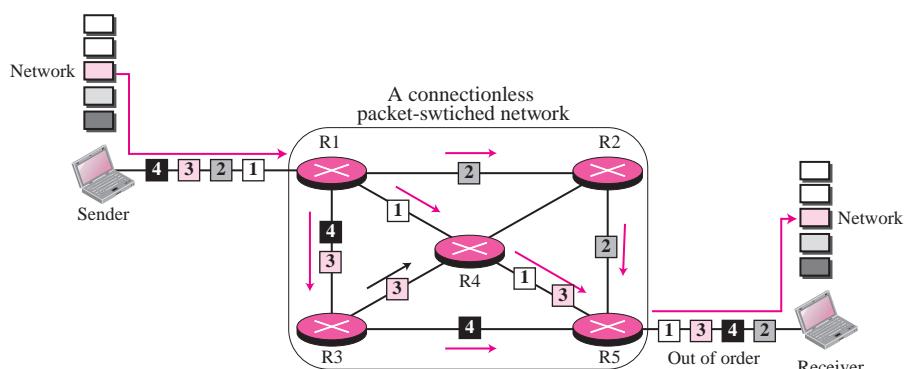
4.3 PACKET SWITCHING AT NETWORK LAYER

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

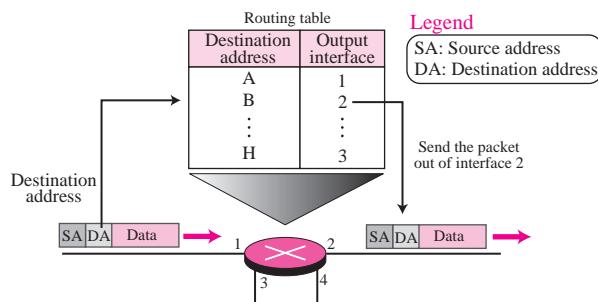


When the network layer provides a connectionless service, each packet traveling in the Internet is an independent entity; there is no relationship between packets belonging to the same message. The switches in this type of network are called *routers*. A packet

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

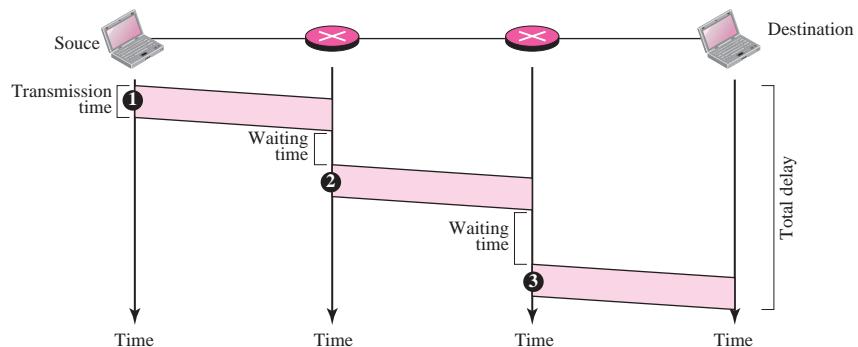


In a connectionless packet-switched network, the forwarding decision is based on the destination address of the packet.

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.

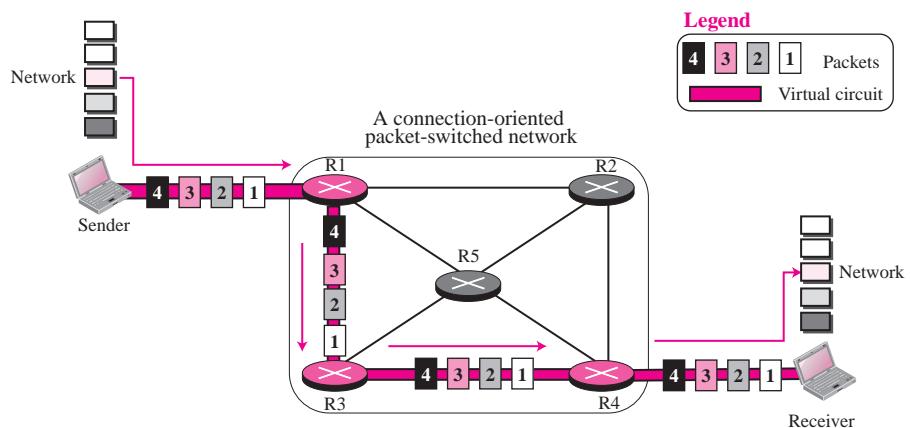
Figure 4.5 Delay in a connectionless network



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

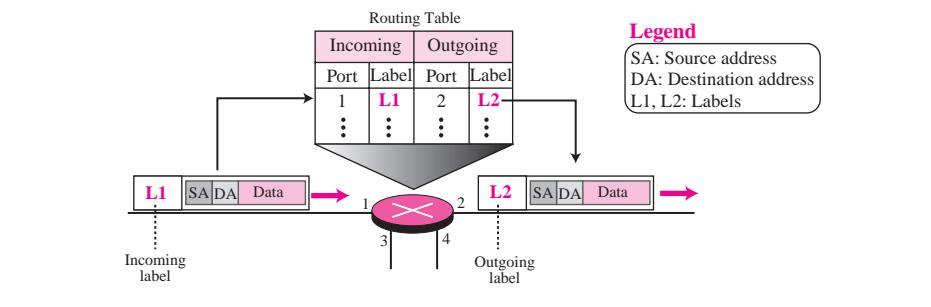


Each packet is forwarded based on the label in the packet. To follow the idea of connection-oriented design to be used in the Internet, we assume that the packet has a label when it reaches the router. Figure 4.7 shows the idea.

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 are 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.

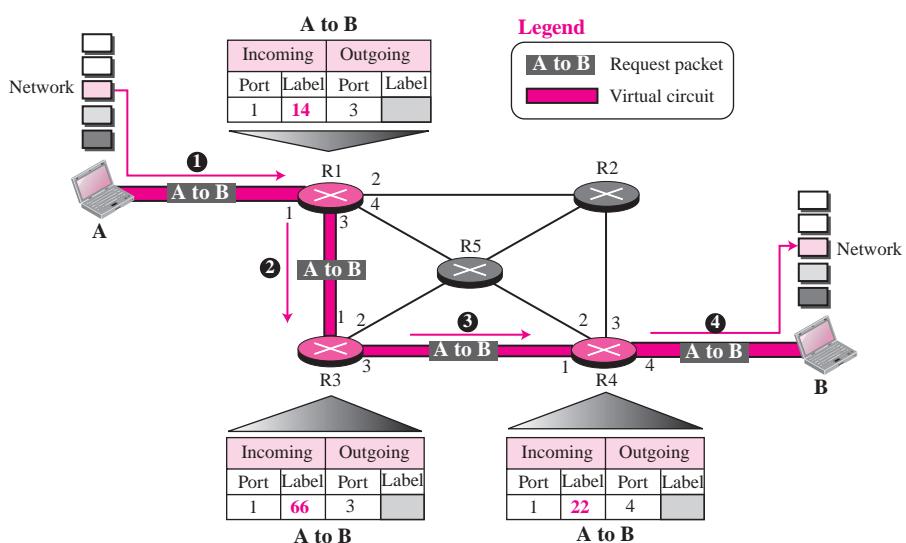
Figure 4.7 Forwarding process in a router when used in a connection-oriented network

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

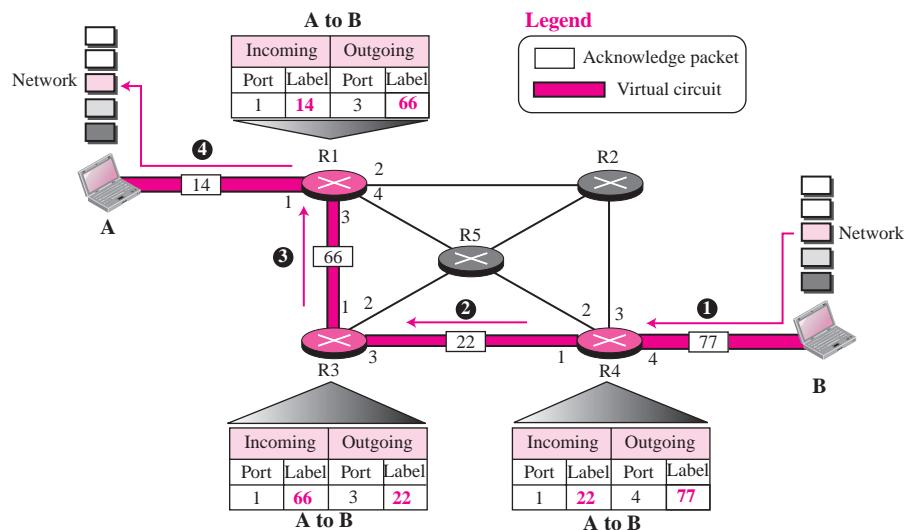
Figure 4.8 Sending request packet in a virtual-circuit network

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



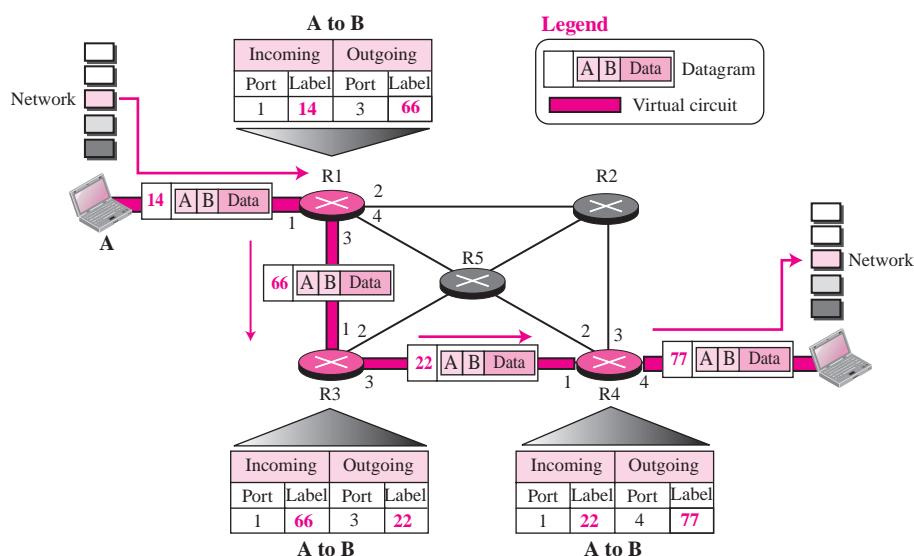
1. The destination sends an acknowledgment to router R4. The acknowledgment carries the global source and destination addresses so the router knows which entry in the table is to be completed. The packet also carries label 77, chosen by the destination as the incoming label for packets from A. Router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



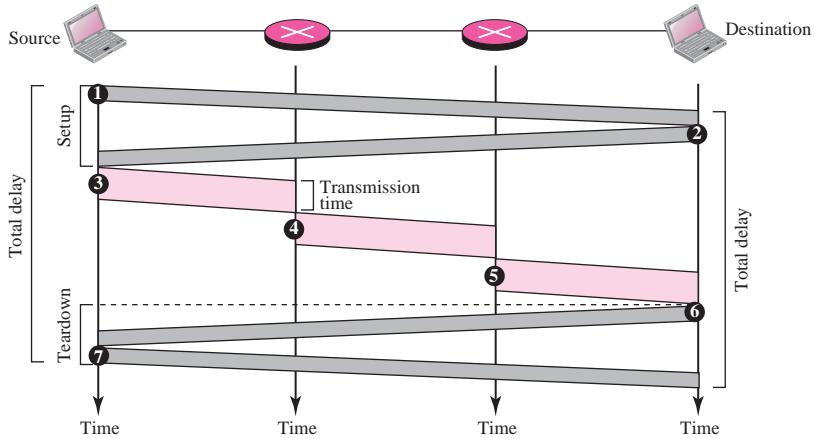
Teardown Phase

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

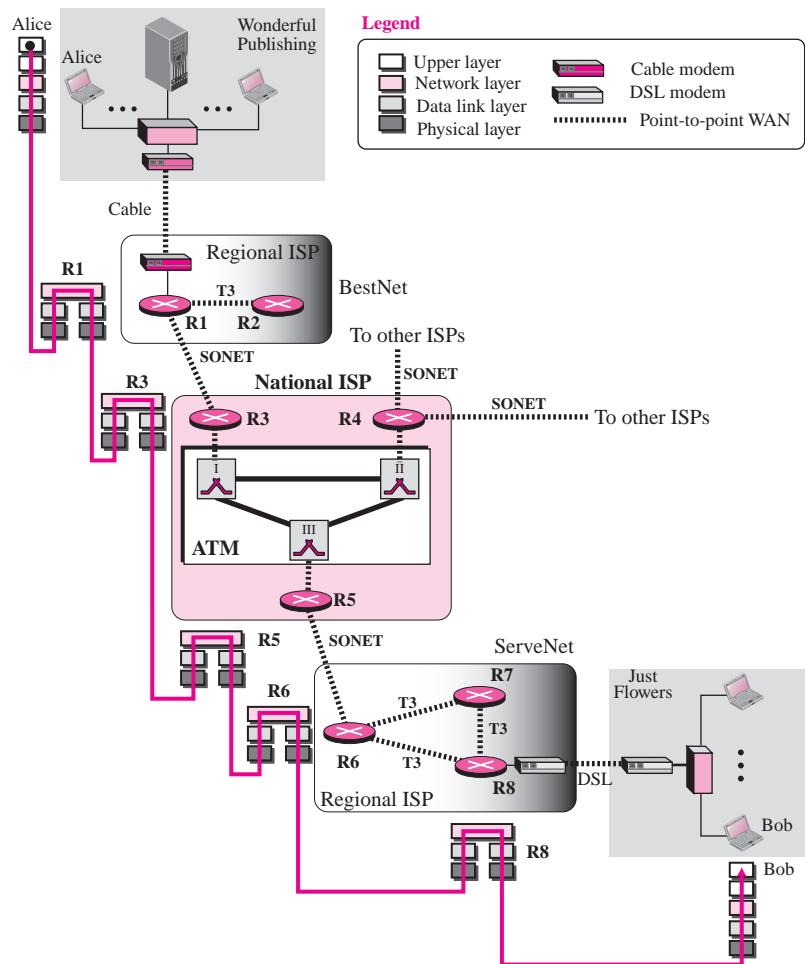


4.4 NETWORK LAYER SERVICES

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

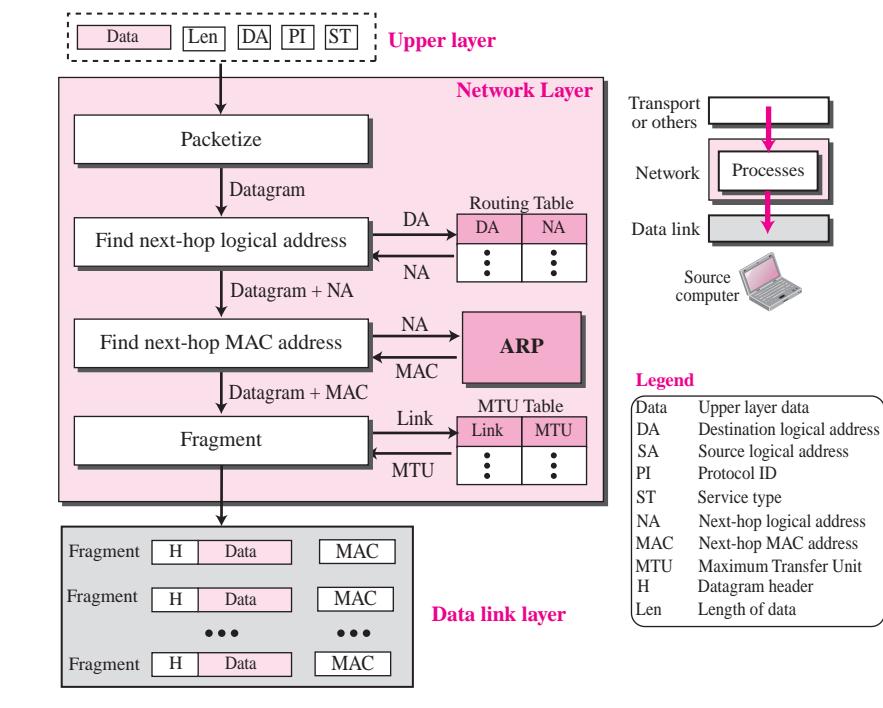
Logical Addressing

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.

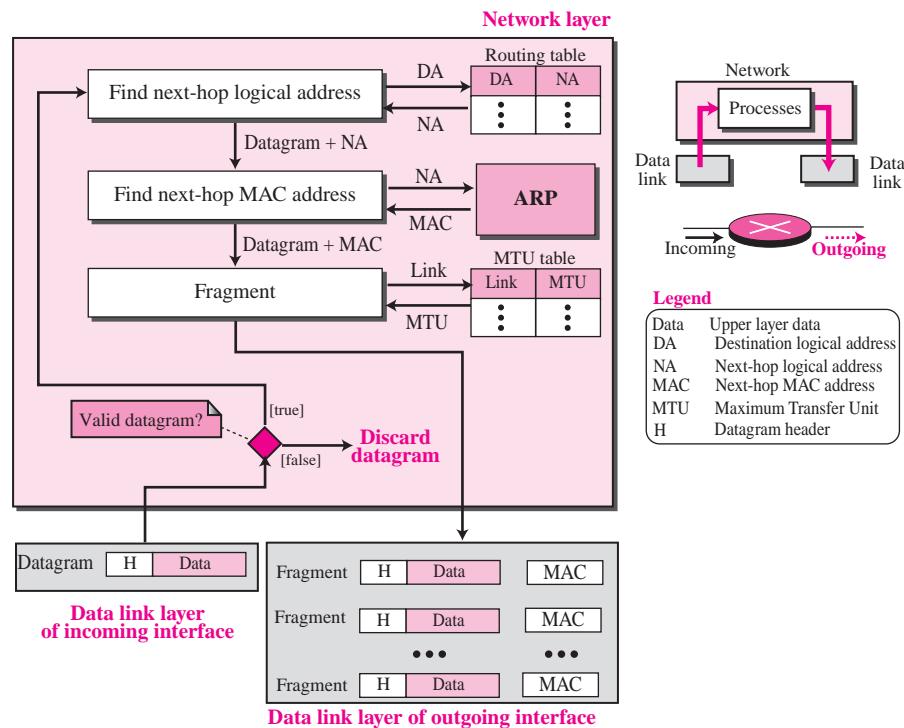
Fragmentation

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 provided by a router at the network layer.

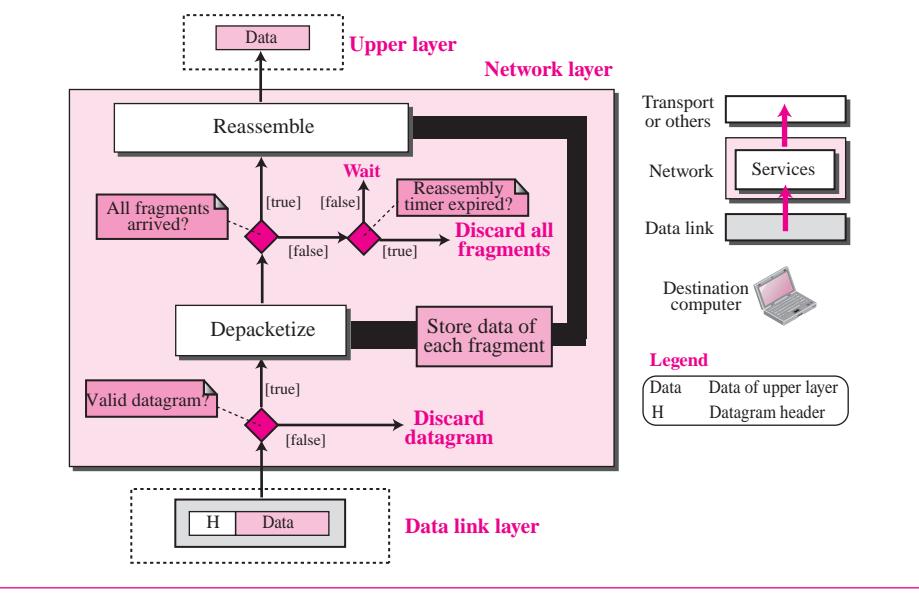
The three processes (finding next-hop logical address, finding next-hop MAC address, and fragmentation) here are the same as the last three processes mentioned for

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

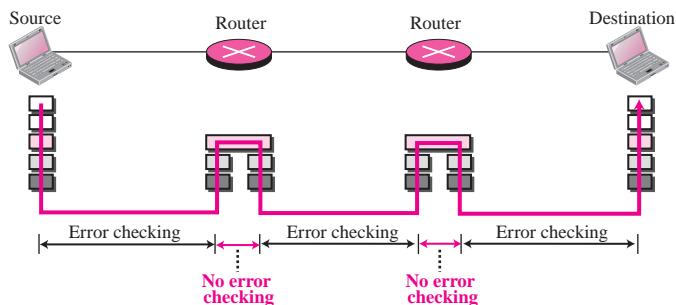
In this section we introduce some issues related to the network layer. These issues actually represent services that are normally discussed for the network layer, but they are either partially implemented at the network layer or not implemented at all. Some services are provided by some auxiliary protocols 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 be 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	flow control
circuit switching	packet switching
congestion control	quality of service (QoS)
connectionless service	routing
connection-oriented service	setup phase
data transfer phase	switching
error control	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?

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

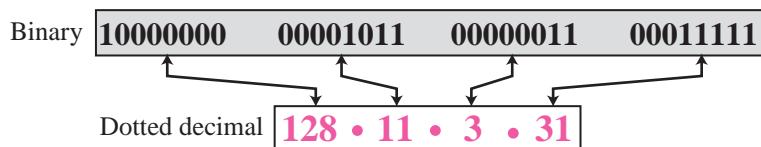
In **binary notation**, an IPv4 address is displayed as 32 bits. To make the address more readable, one or more spaces is usually inserted between each octet (8 bits). Each octet is often referred to as a byte. So it is common to hear an IPv4 address referred to as a 32-bit address, a 4-octet address,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

- 10000001 00001011 00001011 11101111
- 11000001 10000011 00011011 11111111
- 11100111 11011011 10001011 01101111
- 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:

- 129.11.11.239
- 193.131.27.255
- 231.219.139.111
- 249.155.251.15

Example 5.2

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

- 111.56.45.78
- 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 810B0BEF₁₆
- b. 0XC1831BFF or C1831BFF₁₆

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 convert 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.

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

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.

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

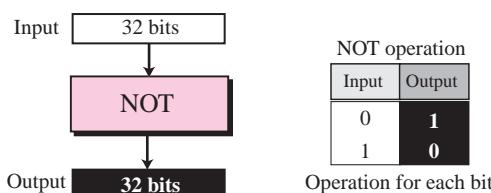
Operations

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

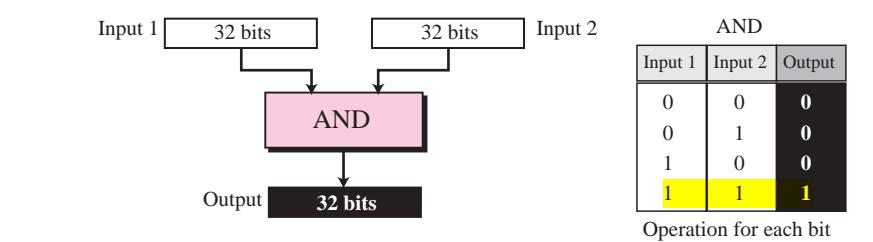
We can use the same operation using the dotted-decimal representation and the short cut.

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

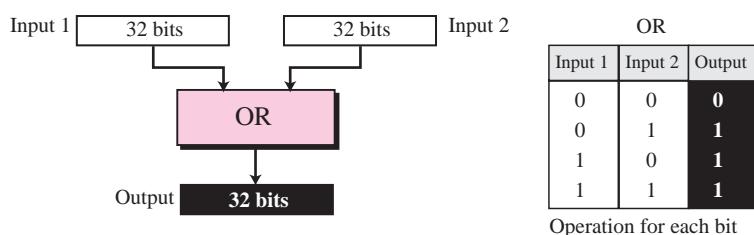
We have applied the first short cut on the first, second, and the fourth byte; we have applied the second short cut on the third byte. We have written **14 and 140** as the sum of terms and selected the smaller term in each pair as shown below.

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



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.

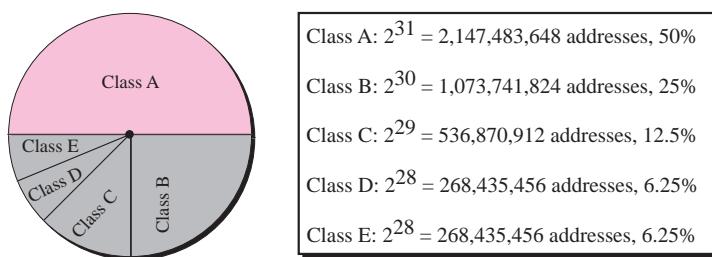
5.2 CLASSFUL ADDRESSING

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).

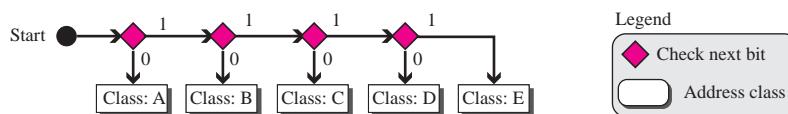
Figure 5.6 Finding the class of an address

	Octet 1	Octet 2	Octet 3	Octet 4		Byte 1	Byte 2	Byte 3	Byte 4
Class A	0.....					0-127			
Class B	10.....					128-191			
Class C	110....					192-223			
Class D	1110....					224-299			
Class E	1111....					240-255			
Binary notation					Dotted-decimal notation				

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:

- 227.12.14.87
- 193.14.56.22
- 14.23.120.8
- 252.5.15.111

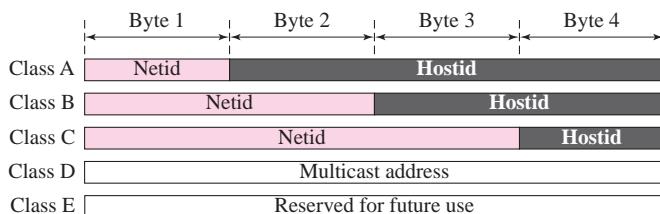
Solution

- The first byte is 227 (between 224 and 239); the class is D.
- The first byte is 193 (between 192 and 223); the class is C.
- The first byte is 14 (between 0 and 127); the class is A.
- 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

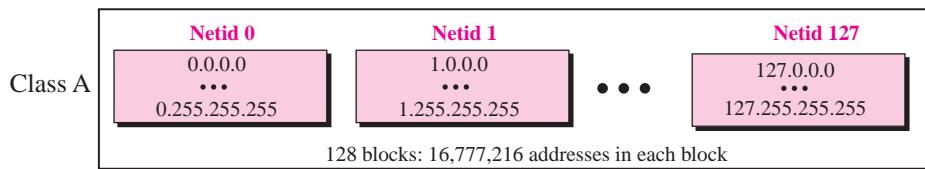
One problem with classful addressing is that each class is divided into a fixed number of blocks with each block having a fixed size. Let us look at each class.

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*

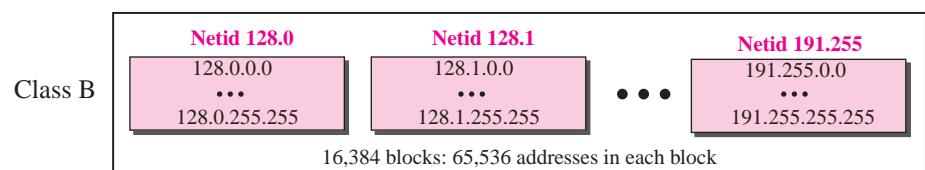


Millions of class A addresses are wasted.

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.

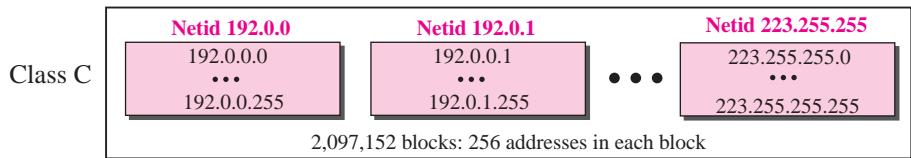
Figure 5.10 *Blocks in class B*



Many class B addresses are wasted.

Class C

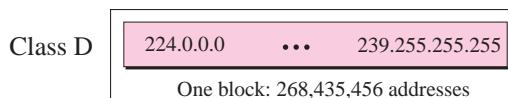
Since 3 bytes in class C define the class and the three leftmost bits should be 110 (fixed), the next 21 bits can be changed to find the number of blocks in this class. Therefore, class C is divided into $2^{21} = 2,097,152$ blocks, in which each block contains 256 addresses, that can be assigned to 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

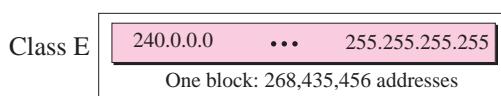
There is just one block of class D addresses. It is designed for **multicasting**, as we will see in a later section. Each address in this class is used to define one group of hosts on the Internet. **When a group is assigned an address in this class, every host that is a member of this group will have 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.

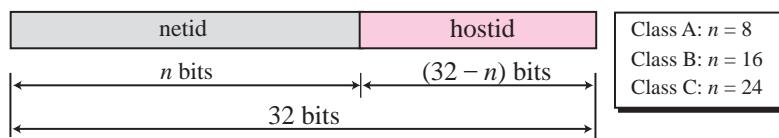
Two-Level Addressing

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.

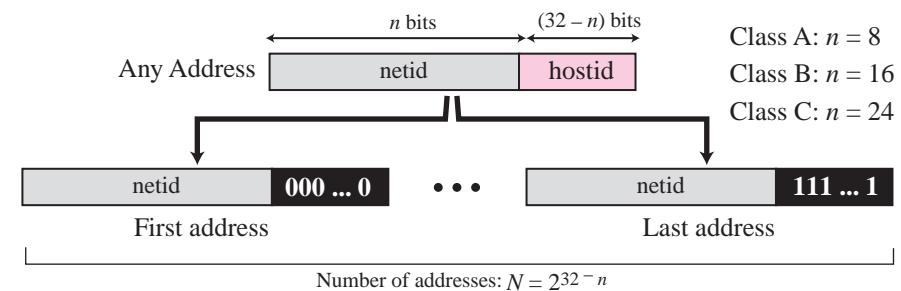
Extracting Information in a Block

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



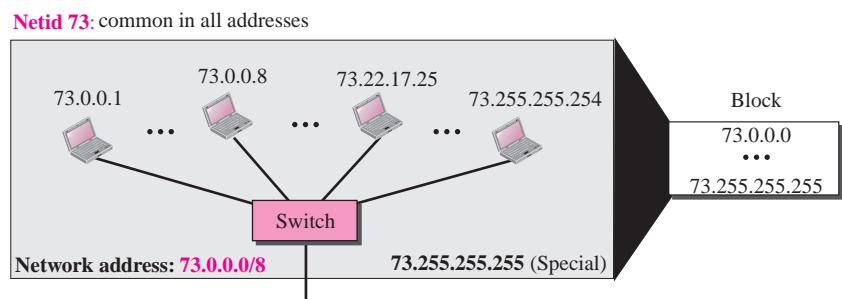
Example 5.13

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.

- 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.

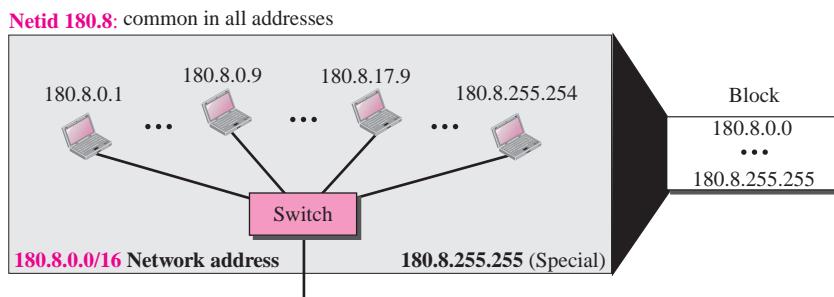
Example 5.14

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



- The number of addresses in this block is $N = 2^{32-n} = 2^{16} = 65,536$.
- 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 .
- 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.

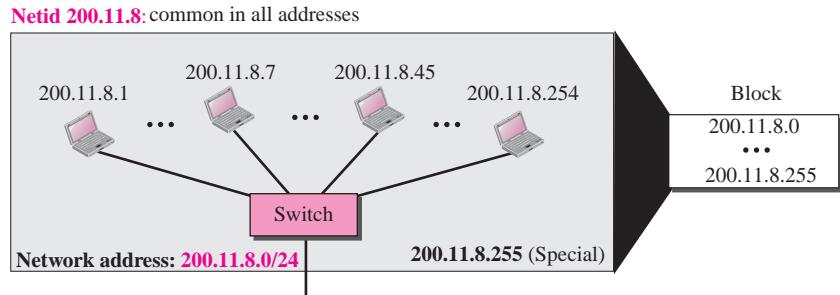
Example 5.15

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

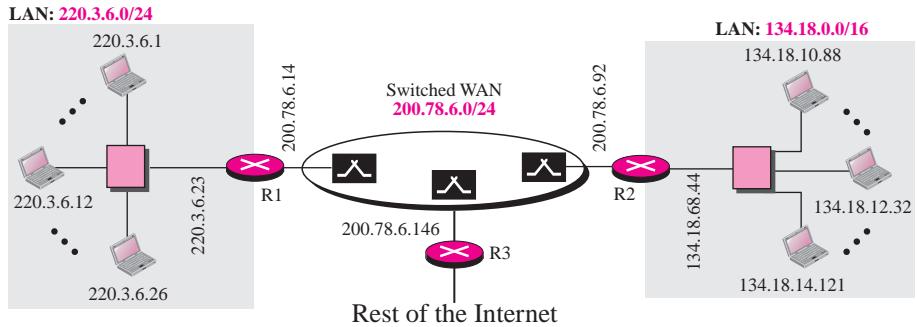
Since 200 is between 192 and 223, the class of the address is C. The value of n for class C is 24. Figure 5.18 shows a possible configuration of the network that uses this block.

- The number of addresses in this block is $N = 2^{32-n} = 2^8 = 256$.
- 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.
- 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.

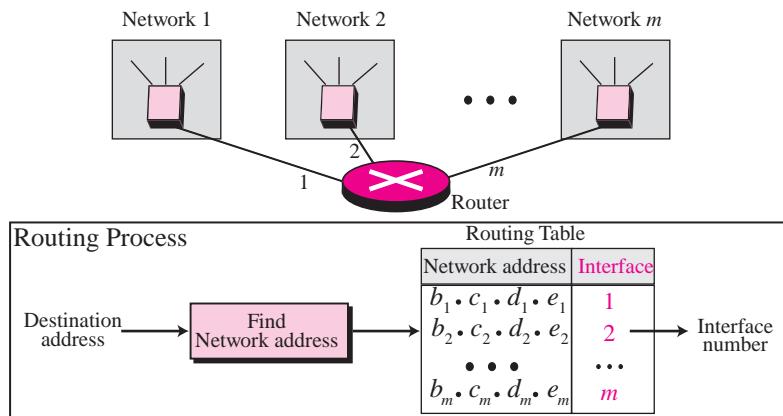
Network Address

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

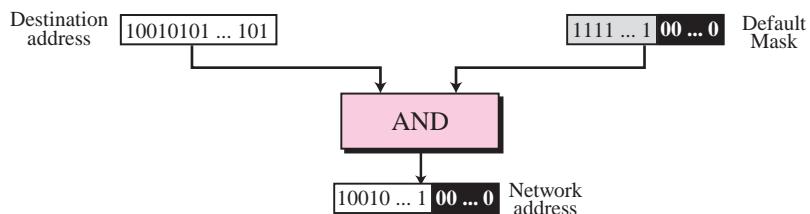
The methods we described previously for extracting the network address are mostly used to show the concept. The routers in the Internet normally use an algorithm to extract the network address from the destination address of a packet. To do this, we need a network mask. A **network mask** or a **default 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

Mask for class A		255.0.0.0
Mask for class B		255.255.0.0
Mask for class C		255.255.255.0

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:

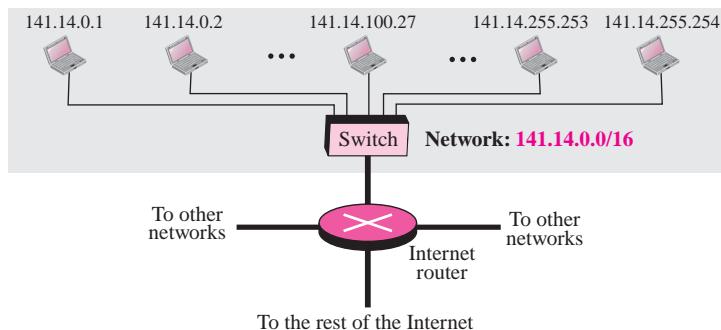
(626) 358 - 1301

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

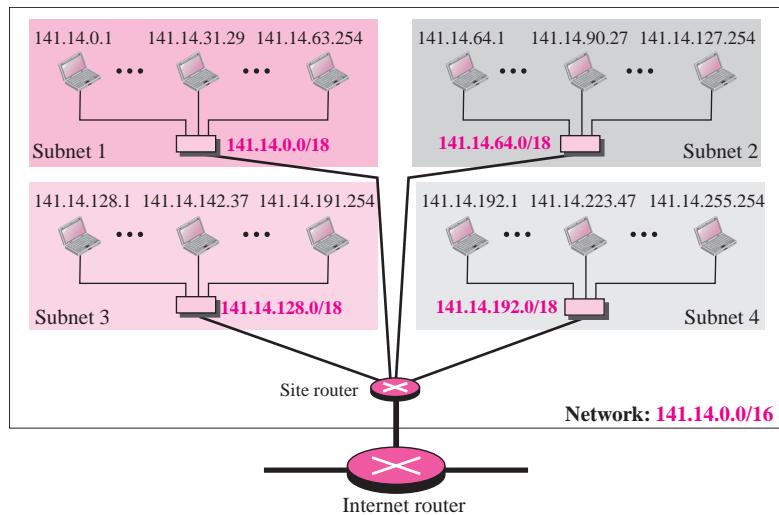
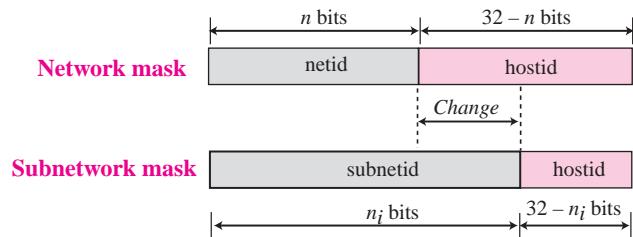


Example 5.19

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

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

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

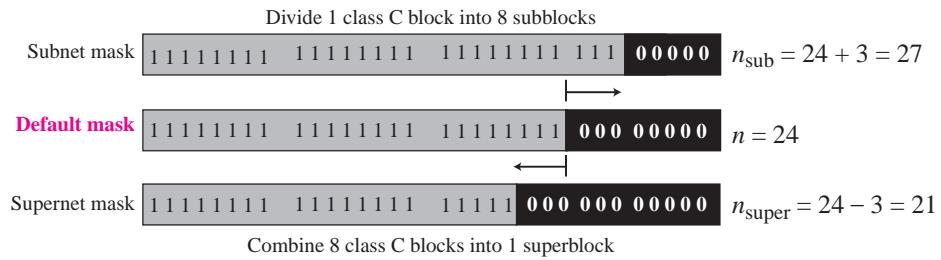
Subnetting could not completely solve address depletion problems in classful addressing because most organizations did not want to share their granted blocks with others. Since class C blocks were still available but the size of the block did not meet the requirement of new organizations that wanted 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 supernet**. 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

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

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.

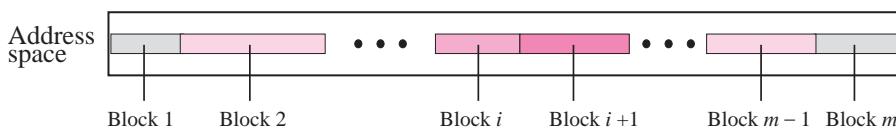
There was another motivation for classless addressing. During the 1990s, Internet service providers (ISPs) came into prominence. An ISP is an organization that provides Internet access for individuals, small businesses, and midsize organizations that do not want to create an Internet site and become 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 $2^0, 2^1, 2^2, \dots, 2^{32}$ 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

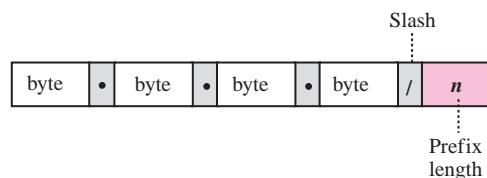
The netid length in classful addressing or the prefix length in classless addressing play a very important role when we need to extract the information about the block from a given address in the block. However, there is a difference here in classful and classless addressing.

- ❑ 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.

- 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:

$$\text{First address} = (\text{any address}) \text{ AND } (\text{network mask})$$

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:

$$\text{Last address} = (\text{any address}) \text{ OR } [\text{NOT } (\text{network mask})]$$

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 $2^{32-n} = 2^{32-27} = 2^5 = 32$.
- 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:

$$n_{\text{sub}} = n + \log_2 (N/N_{\text{sub}})$$

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

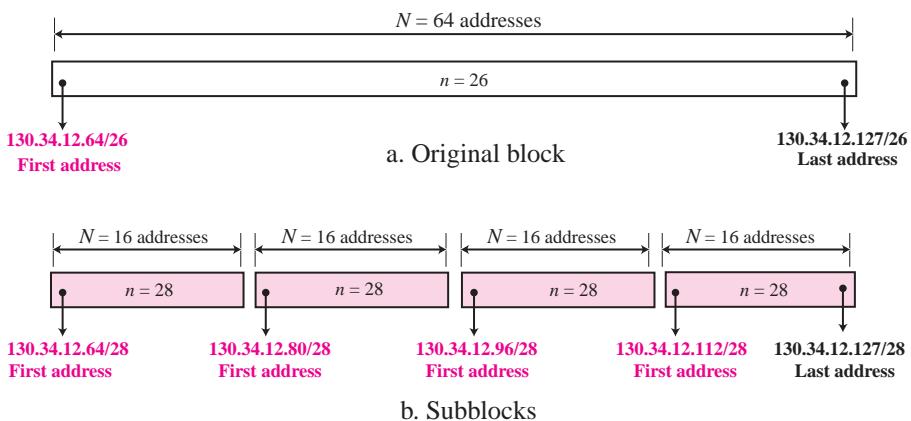
The number of addresses for the whole network can be found as $N = 2^{32-26} = 64$. Using the process described in the previous section, the first address in the network is 130.34.12.64/26 and the last address is 130.34.12.127/26. We now design the subnetworks:

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.

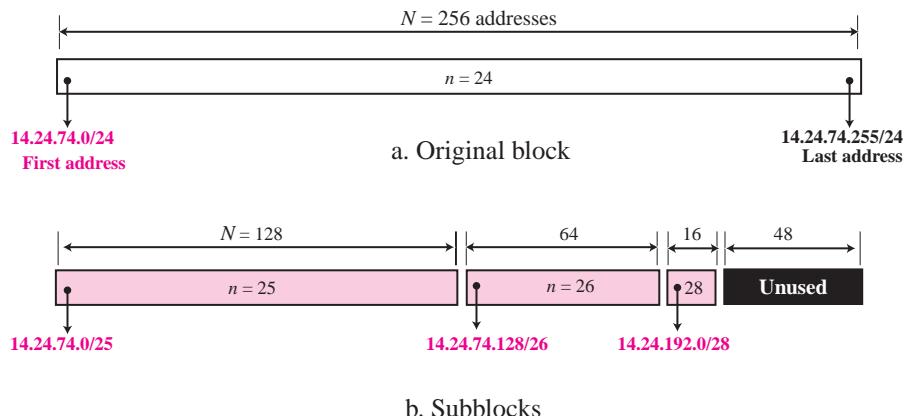
Solution

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.

- 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



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:

$$\text{Central office } N_c = 32$$

$$\text{East office } N_e = 16$$

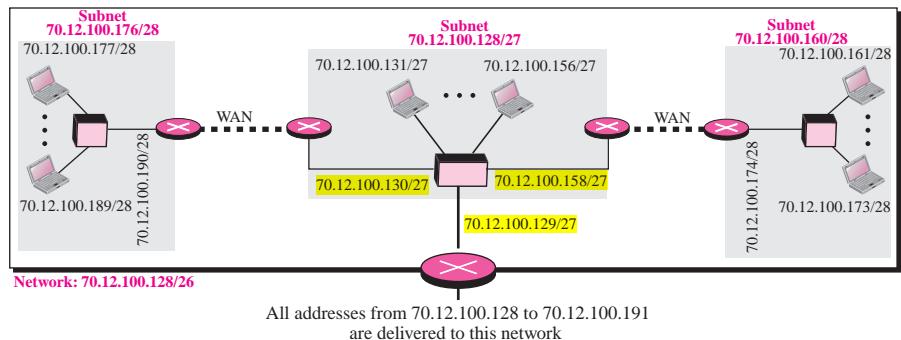
$$\text{West office } N_w = 16$$

2. We can find the prefix length for each subnetwork:

$$n_c = n + \log_2(64/32) = 27 \quad n_e = n + \log_2(64/16) = 28 \quad n_w = n + \log_2(64/16) = 28$$

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



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.

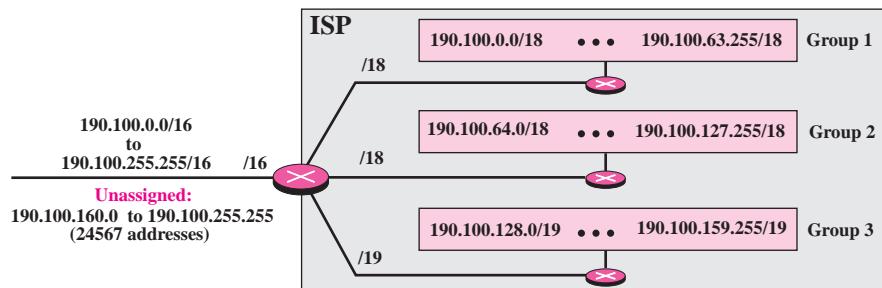
Solution

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 be 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: $128 \times 64 = 8192$	$n_3 = 16 + \log_2 (65536/8192) = 19$

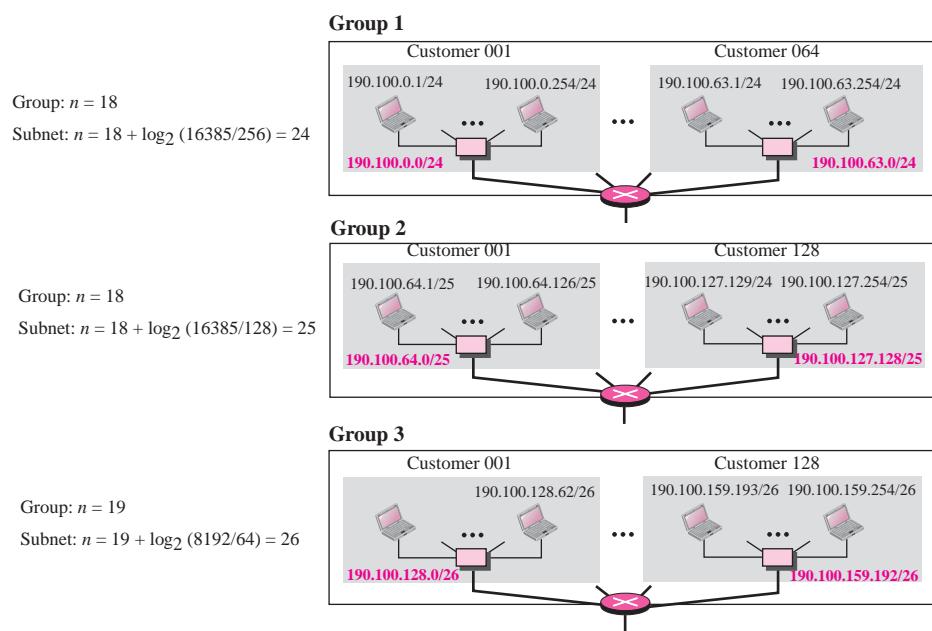
Figure 5.33 shows the design for the first hierarchical level.

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.

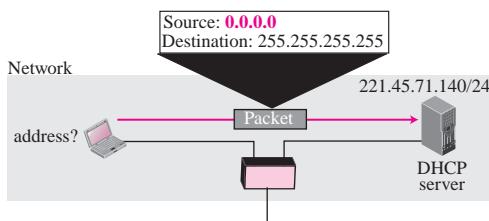
Special Blocks

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

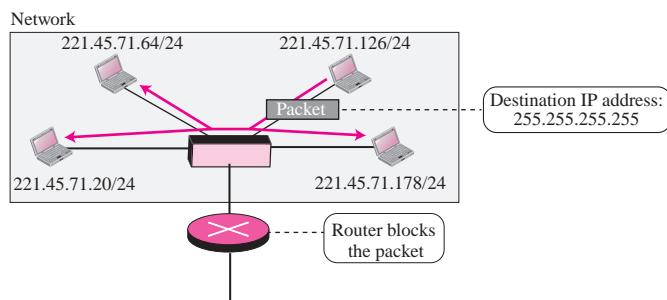
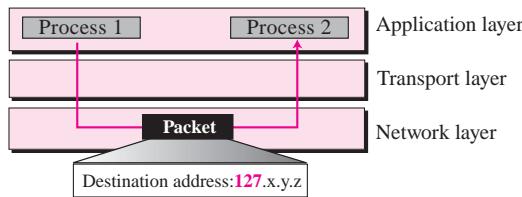


All-Ones Address: Limited Broadcast Address

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

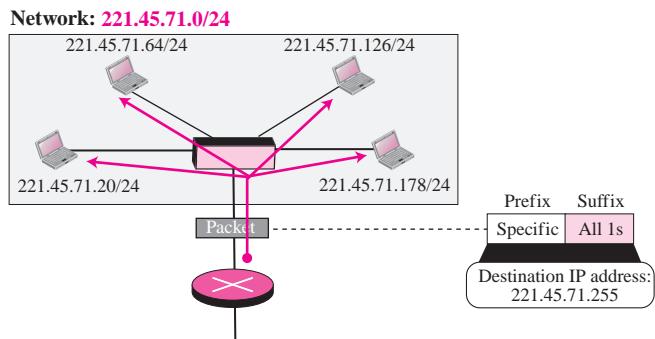
We have already discussed network addresses. The first address (with the suffix set all to 0s) in a block defines the network address. It actually defines the network itself

(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



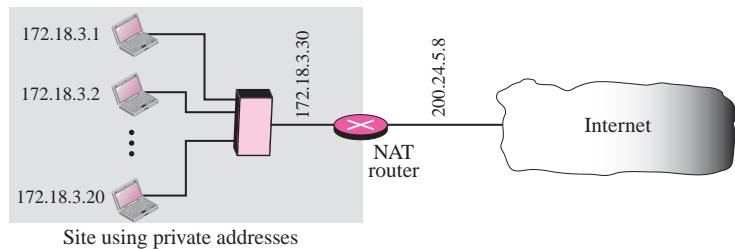
5.5 NAT

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

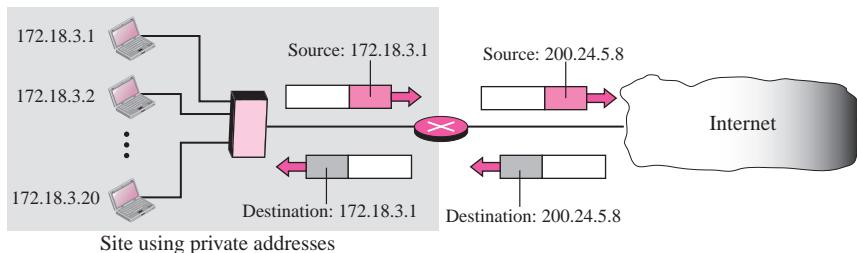


As the figure shows, the private network uses private addresses. The router that connects the network to the global address uses one private address and one global address. The private network is transparent to the rest of the Internet; the rest of the Internet sees only the NAT router with the address 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

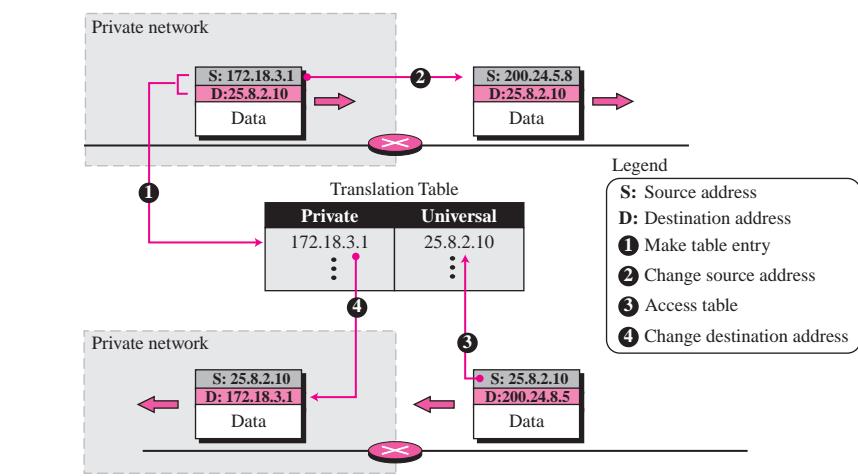
The reader may have noticed that translating the source addresses for an outgoing packet is straightforward. But how does the NAT router know the destination address for a packet coming from the Internet? There may be tens or hundreds of private IP

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	limited broadcast address
address space	loopback address
binary notation	netid
block of addresses	network address
class A address	Network Address Translation (NAT)
class B address	network mask
class C address	prefix
class D address	prefix length
class E address	slash notation
classful addressing	subnet
classless addressing	subnet mask
classless interdomain routing (CIDR)	subnetting
default mask	subnetwork
direct broadcast address	suffix
dotted-decimal notation	suffix length
hexadecimal notation	supernet mask
hostid	supernetting
IP address	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.
- Find the subnet mask.
 - Find the number of addresses in each subnet.
 - Find the first and the last address in the first subnet.
 - 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.
- Find the subnet mask.
 - Find the number of addresses in each subnet.
 - Find the first and the last address in the first subnet.
 - 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.
- Find the subnet mask.
 - Find the number of addresses in each subnet.
 - Find the first and the last address in the first subnet.
 - Find the first and the last address in the last subnet (subnet 32).
- 27.** Write the following mask in slash notation (/n):
- 255.255.255.0
 - 255.0.0.0
 - 255.255.224.0
 - 255.255.240.0
- 28.** Find the range of addresses in the following blocks:
- 123.56.77.32/29
 - 200.17.21.128/27
 - 17.34.16.0/23
 - 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:
- 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.

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.

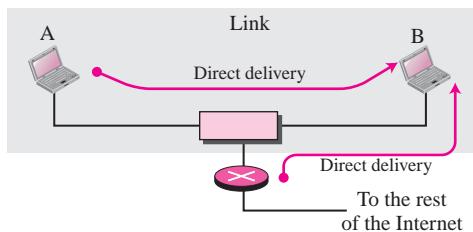
6.1 DELIVERY

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.

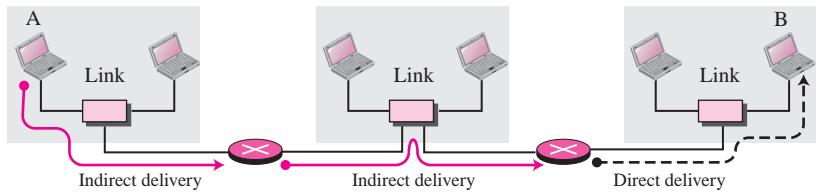
In direct delivery, the sender uses the destination IP address to find the destination physical address. The IP software then gives the destination IP address with the destination physical address to the data link layer for actual delivery. This process is called *mapping the IP address to the physical 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

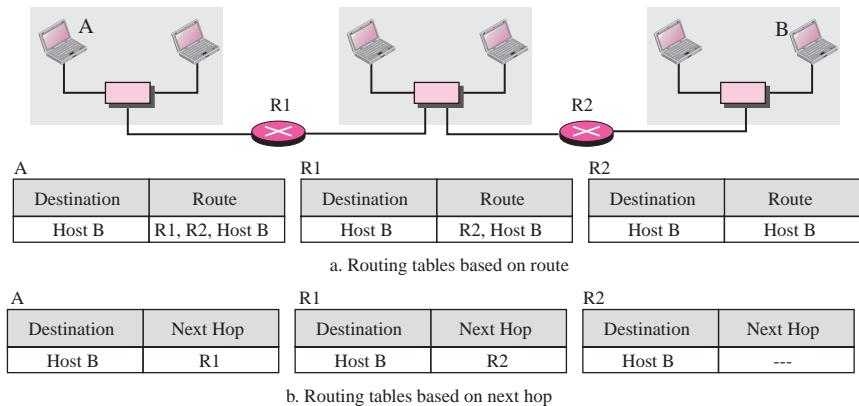
We first discuss forwarding based on the destination address. This is a traditional approach, which is prevalent today. In this case, forwarding requires a host or a router to have a routing table. When a host has a packet to send or when a router has received a packet to be forwarded, it looks at this 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.

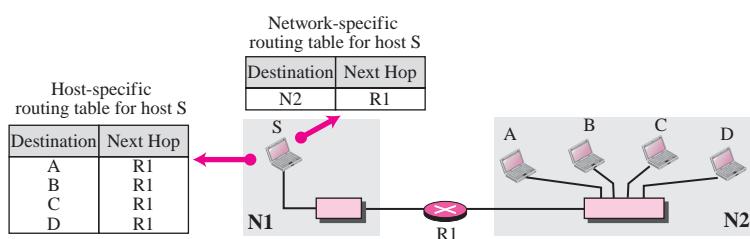
Next-Hop Method One technique to reduce the contents of a routing table is called the **next-hop method**. In this technique, the routing table holds only the address of the next hop instead of information about the complete route. The entries of a routing table must be consistent with each other. Figure 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*

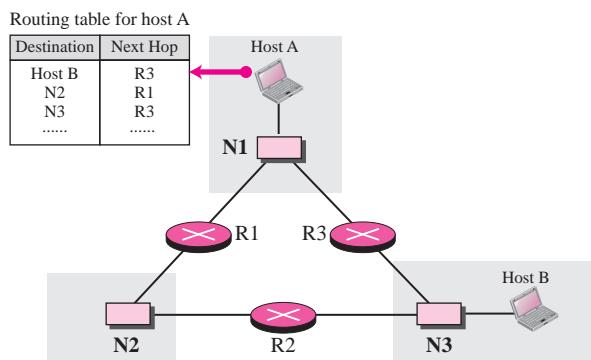


Host-Specific Method In the **host-specific method**, the destination host address is given in the routing table. The rationale behind this method is the inverse of the network-specific method. Here efficiency is sacrificed for other advantages: Although it is not efficient to put the host address in 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.

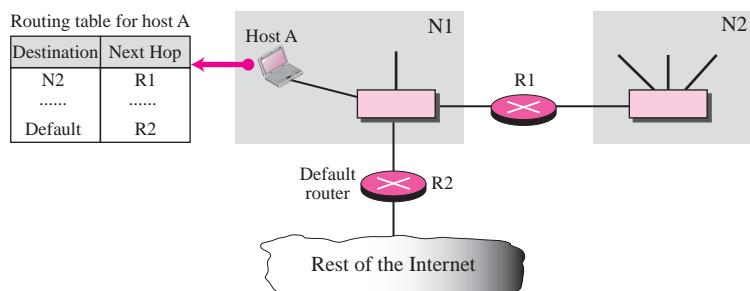
Host-specific routing is used for purposes such as checking the route or providing security measures.

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

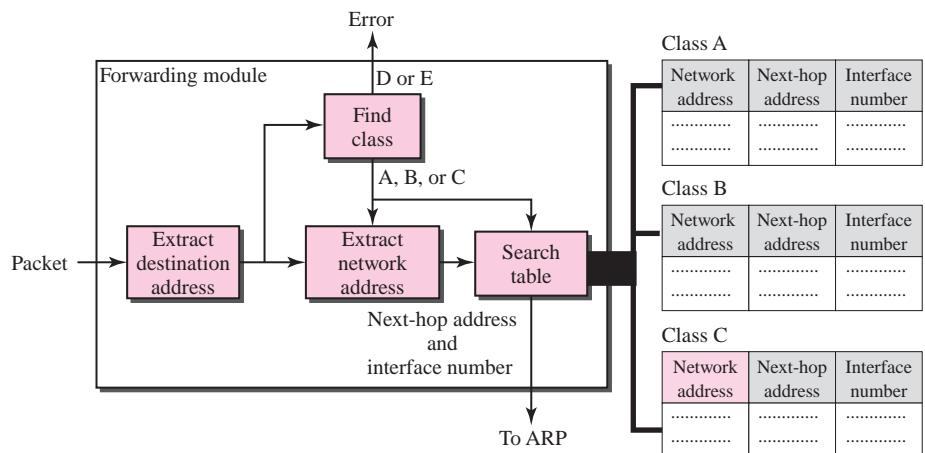
and forwarding module for the situation in which there is no subnetting. We then show how the module changes if subnetting is involved.

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:

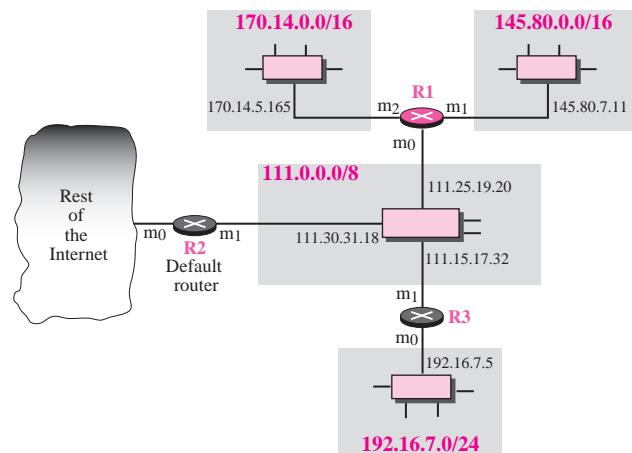
1. The destination address of the packet is extracted.
2. A copy of the destination address is used to find the class of the address. This is done by shifting the copy of the address 28 bits to the right. The result is a 4-bit number between 0 and 15. If the result is
 - a. 0 to 7, the class is A.
 - 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 right-most 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 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 00001**100** or 12. The destination network is class C. The network address is extracted by masking off the left-most 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 00001**010** 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).

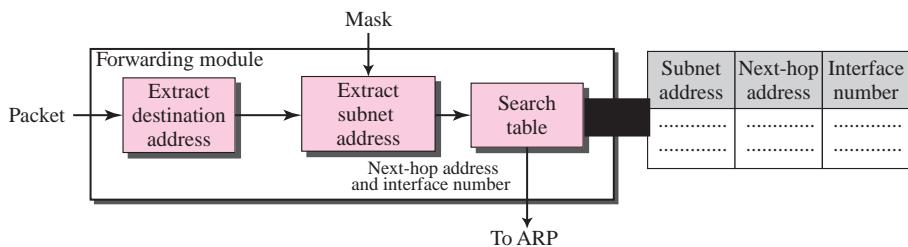
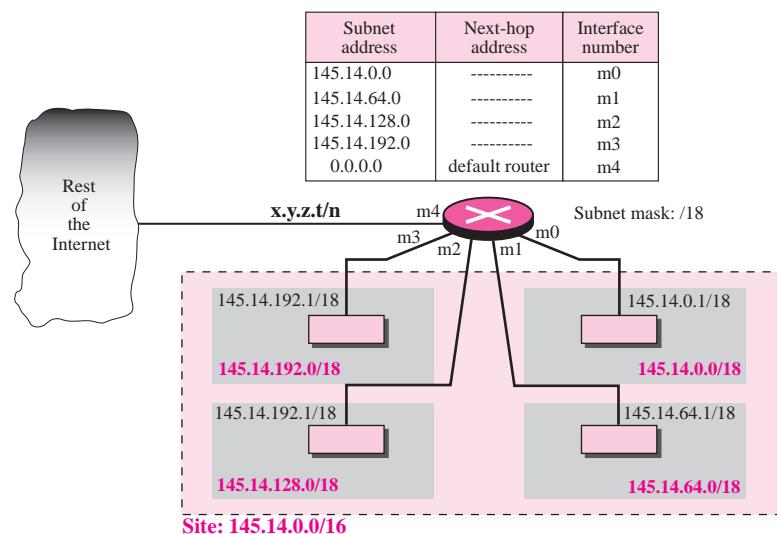
Figure 6.10 Simplified forwarding module in classful address with subnetting**Example 6.4**

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

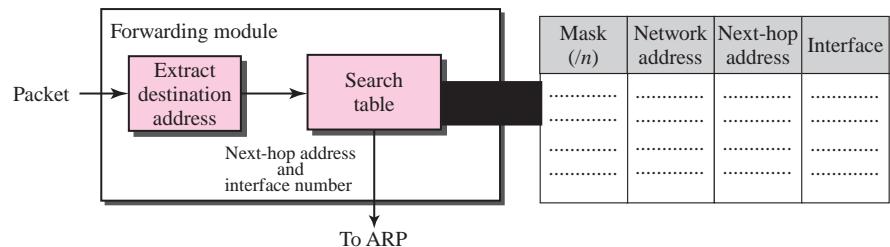
In classless addressing, the whole address space is one entity; there are no classes. This means that forwarding requires one row of information for each block involved. The table needs to be searched based on the network address (first address in the block). Unfortunately, the destination address in 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.

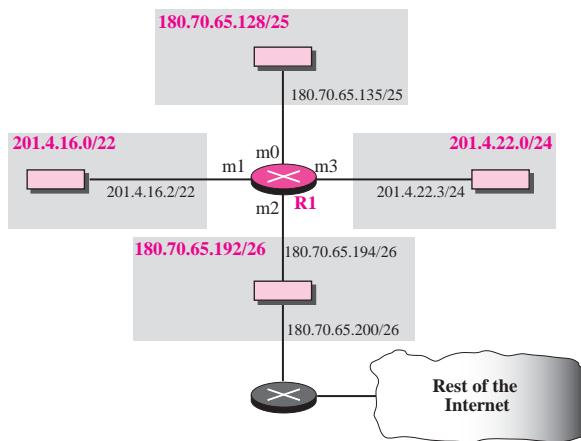
Figure 6.13 Configuration for Example 6.7**Solution**

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

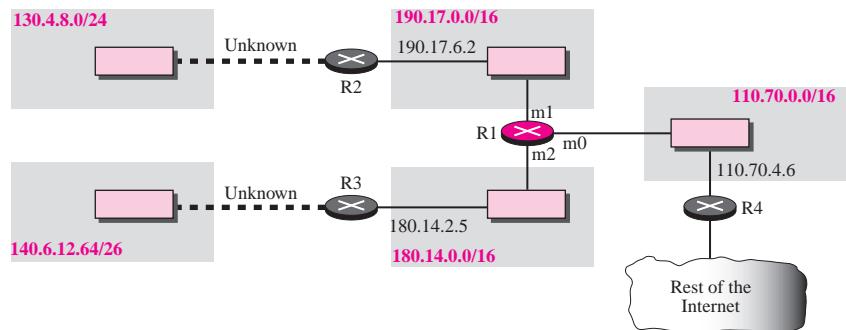
Now let us give a different type of example. Can we find the configuration of a router if we know only its routing table? The routing table for router R1 is given in Table 6.2. Can we draw its topology?

Table 6.2 Routing table for Example 6.11

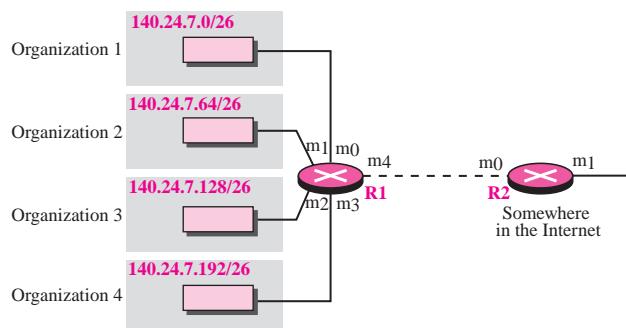
Mask	Network Address	Next-Hop Address	Interface Number
/26	140.6.12.64	180.14.2.5	m2
/24	130.4.8.0	190.17.6.2	m1
/16	110.70.0.0	-----	m0
/16	180.14.0.0	-----	m2
/16	190.17.0.0	-----	m1
Default	Default	110.70.4.6	m0

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

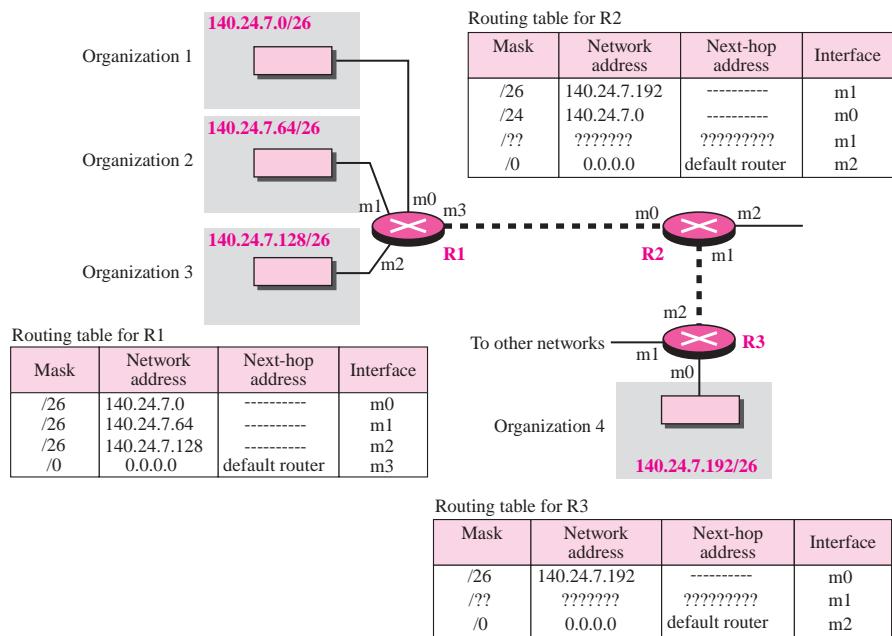
R1 is connected to networks of four organizations that each use 64 addresses. R2 is somewhere far from R1. R1 has a longer routing table because each packet must be correctly routed to the appropriate organization. R2, on the other hand, can have a very

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



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.

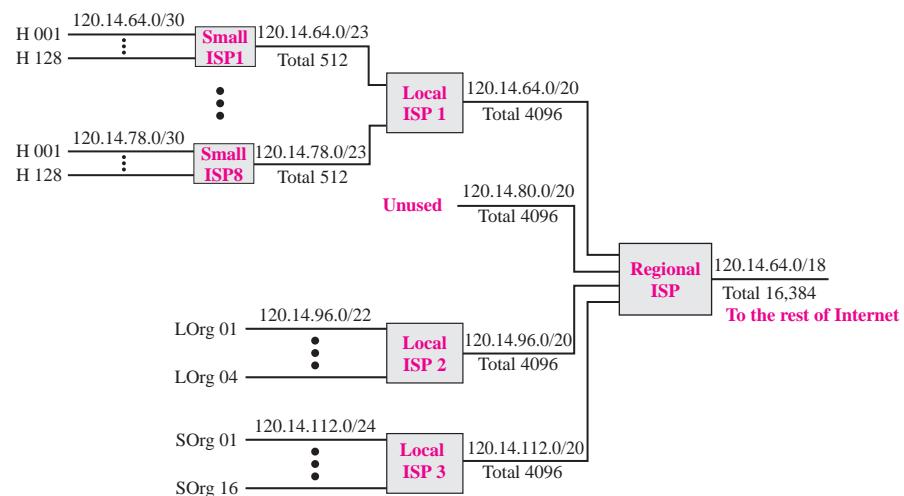
Let us take the case of a local ISP. A local ISP can be assigned a single, but large, block of addresses with a certain prefix length. The local ISP can divide this block into smaller blocks of different sizes, and assign these to individual users and organizations, both large and small. If the block 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 ISPs, and one is left unused.

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.

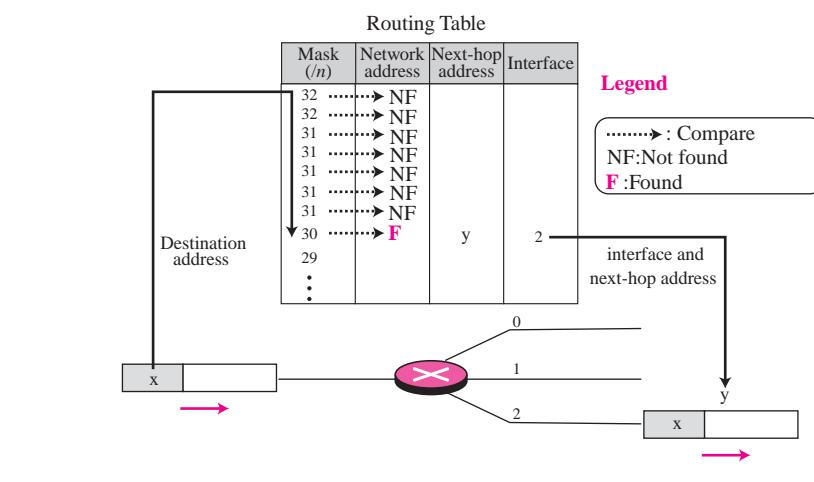
Forwarding Based on Label

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



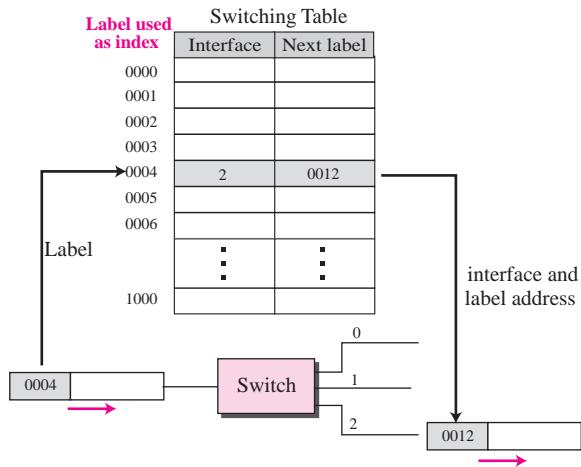
When the forwarding algorithm gets the destination address of the packet, it needs to delve into the mask column. For each entry, it needs to apply the mask to find the destination network address. It then needs to check the network addresses in the table until it finds the match. The router then extracts 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

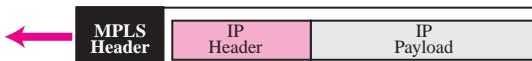
During the 1980s, several vendors created routers that implement switching technology. Later IETF approved a standard that is called Multi-Protocol Label Switching. In

Figure 6.19 Example 6.14: Forwarding based on label

in 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

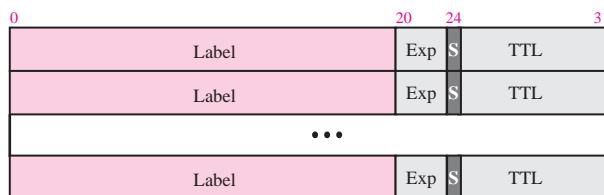
To simulate connection-oriented switching using a protocol like IP, the first thing that is needed is to add a field to the packet that carry the label discussed in Chapter 4. The IPv4 packet format does not allow this extension (although this field is provided in IPv6 packet format, as we will see in 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

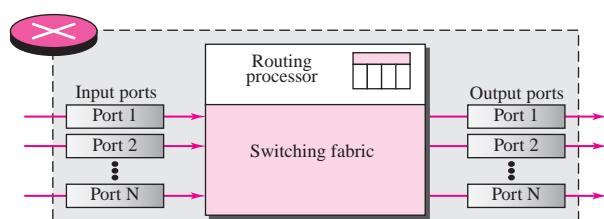
A stack of labels in MPLS allows hierarchical switching. This is similar to conventional hierarchical routing. For example, a packet with two labels can use the top label to forward the packet through switches outside an organization; the bottom label can be used to route the packet inside the organization 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

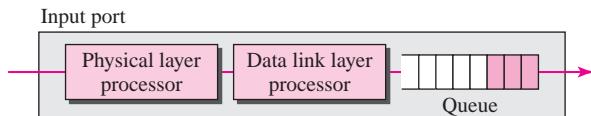
We can say that a router has four components: **input ports**, **output ports**, the **routing processor**, and the **switching fabric**, as shown in Figure 6.22.

Figure 6.22 Router components

Input Ports

Figure 6.23 shows a schematic diagram of an input port.

Figure 6.23 *Input port*

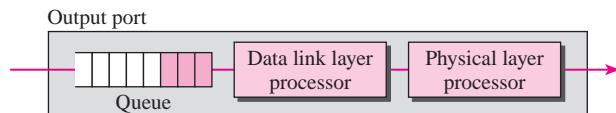


An input port performs the physical and data link layer functions of the router. The bits are constructed from the received signal. The packet is decapsulated from the frame. Errors are detected and corrected. The packet is ready to be forwarded by the network layer. In addition to a physical layer processor 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.

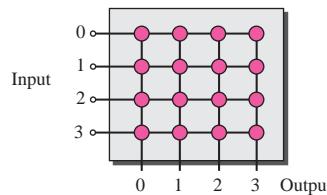
Switching Fabrics

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.

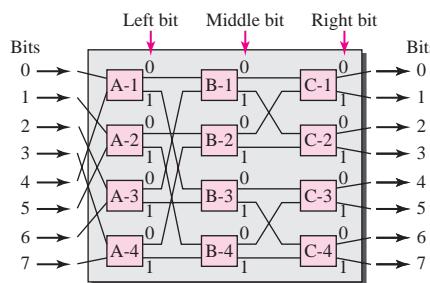
Crossbar Switch The simplest type of switching fabric is the crossbar switch shown in Figure 6.25. A **crossbar switch** connects n inputs to n outputs in a grid, using electronic microswitches at each **crosspoint**.

Figure 6.25 *Crossbar switch*

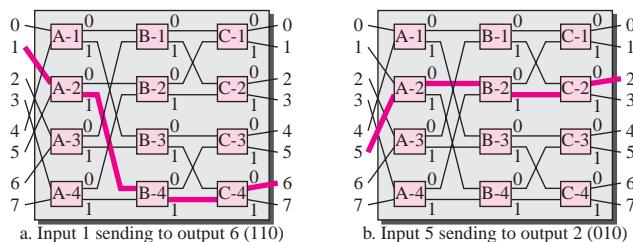


Banyan Switch More realistic than the crossbar switch is the **banyan switch** (named after the banyan tree) as shown in Figure 6.26.

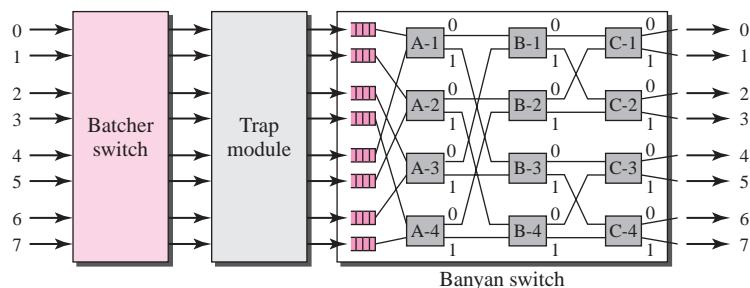
Figure 6.26 *A banyan switch*



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

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	indirect delivery
Batcher-banyan switch	input ports
crossbar switch	longest mask matching
crosspoint	longest prefix match
default method	network-specific method
delivery	next-hop address
direct delivery	next-hop method
forwarding	output ports
host-specific method	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.

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

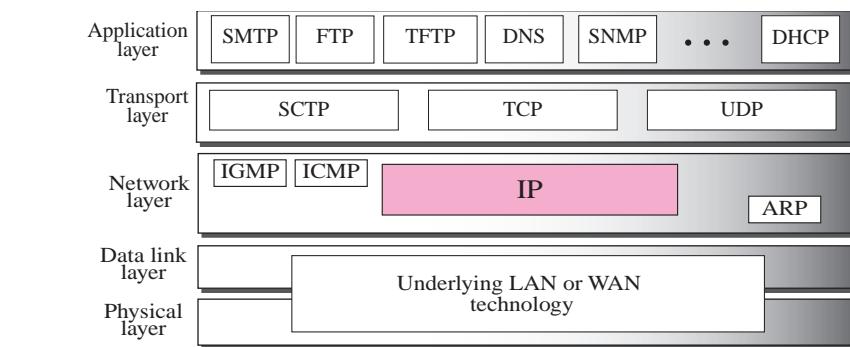
The chapter has several objectives:

- ❑ 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.

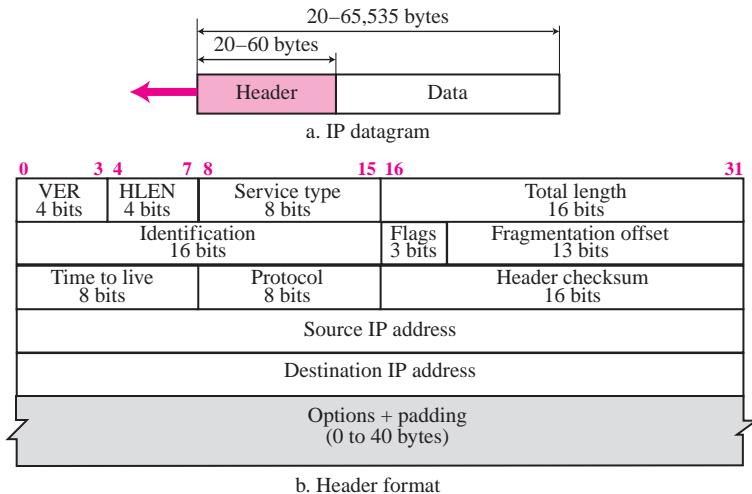
IP is also a connectionless protocol for a packet switching network that uses the datagram approach (see Chapter 4). This means that each datagram is handled independently, and each datagram can follow a different route to the destination. This implies that datagrams sent by the same source to the same 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.

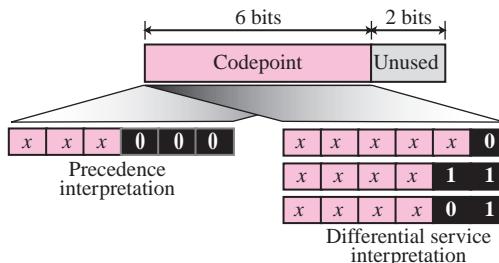
Figure 7.2 IP datagram



- ❑ **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.

$$\text{Length of data} = \text{total length} - \text{header length}$$

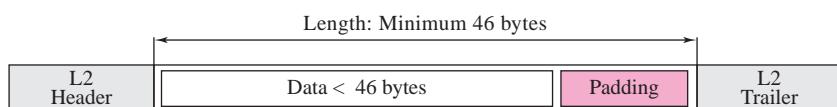
Since the field length is 16 bits, the total length of the IP datagram is limited to $65,535 (2^{16} - 1)$ bytes, of which 20 to 60 bytes are the header and the rest is data from the upper layer.

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.

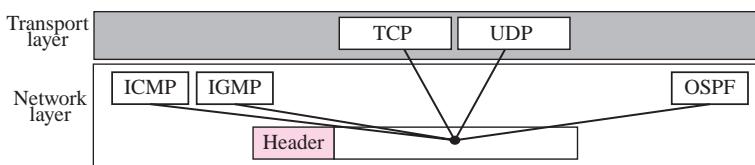
This field is needed because routing tables in the Internet can become corrupted. A datagram may travel between two or more routers for a long time without ever getting delivered to the destination host. This field limits the lifetime of a datagram.

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 8×4 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 5×4 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

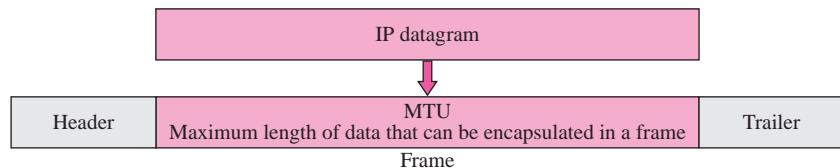
To find the time-to-live field, we skip 8 bytes (16 hexadecimal digits). The time-to-live field is the ninth byte, which is 01. This means the packet can travel only one hop. The protocol field is the next byte (02), which means that the upper layer protocol is IGMP (see Table 7.2).

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).

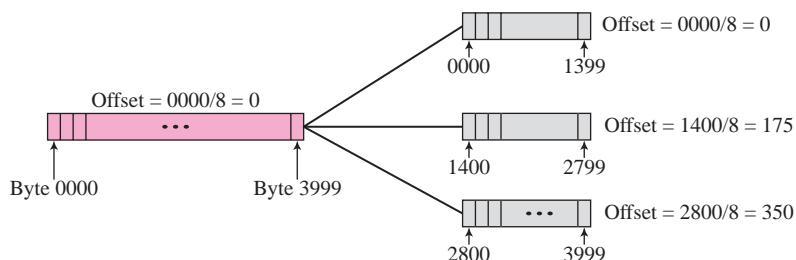
Figure 7.7 Flags field

D: Do not fragment
M: More fragments



- ❑ **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 $2800/8 = 350$.

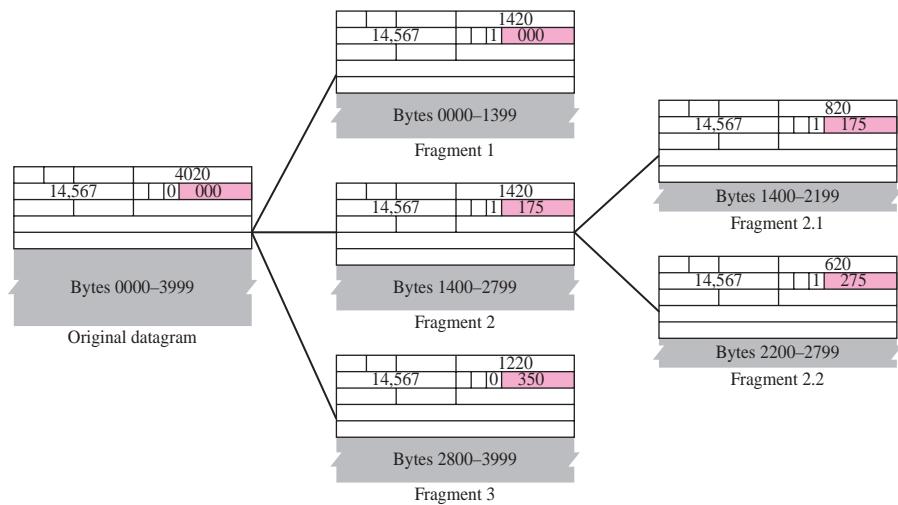
Figure 7.8 Fragmentation example



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:

- The first fragment has an offset field value of zero.
- Divide the length of the first fragment by 8. The second fragment has an offset value equal to that result.
- Divide the total length of the first and second fragment by 8. The third fragment has an offset value equal to that result.
- 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 (5×4), 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

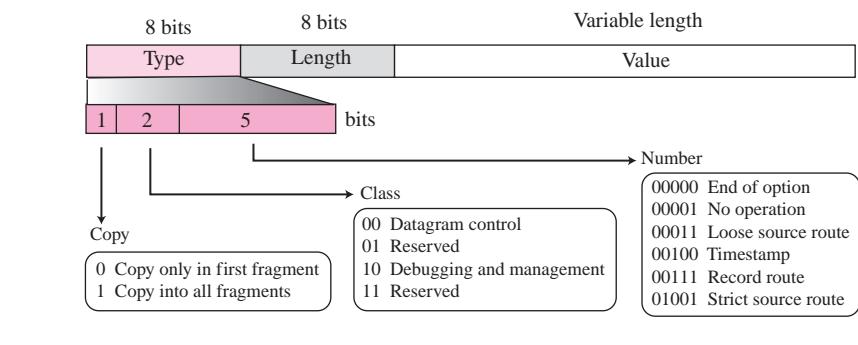
The header of the IP datagram is made of two parts: a fixed part and a variable part. The fixed part is 20 bytes long and was discussed in the previous section. The variable part comprises the options, which can be a maximum of 40 bytes.

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.

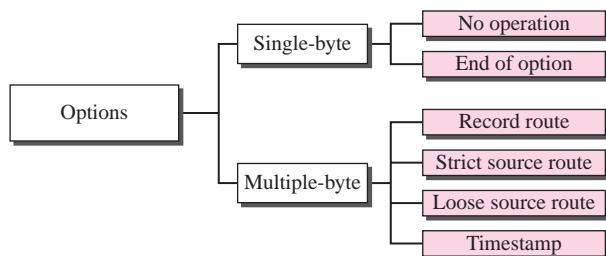
Value

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).

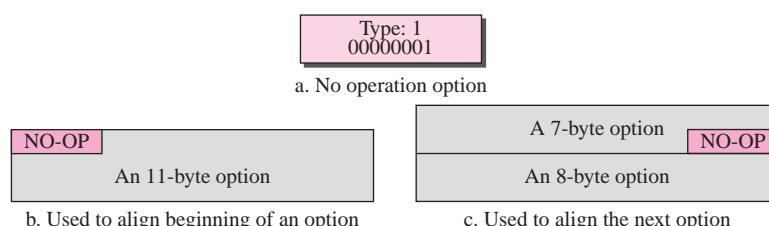
Figure 7.11 Categories of options



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

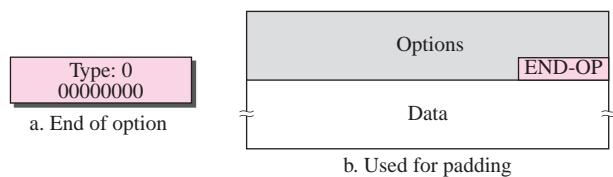


End-of-Option Option

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).

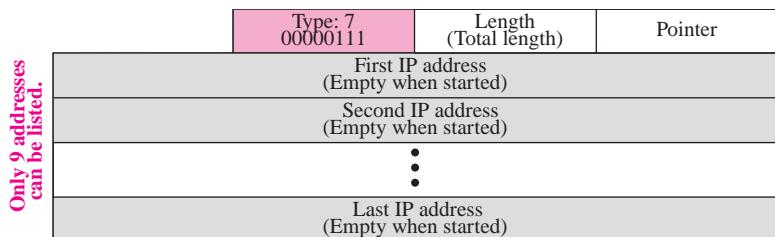
Figure 7.13 End-of-option option



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.

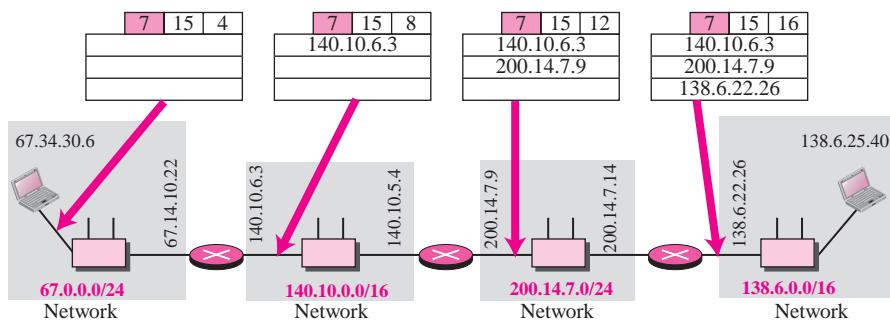
Figure 7.14 Record-route option



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

Type: 137 10001001	Length (Total length)	Pointer
First IP address (Filled when started)		
Second IP address (Filled when started)		
⋮		
Last IP address (Filled when started)		

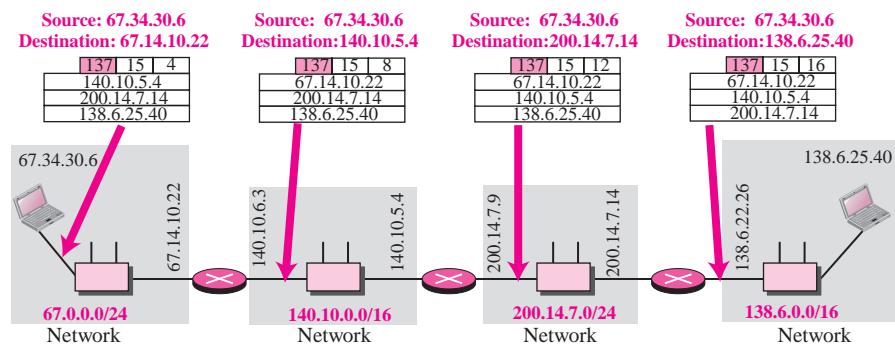
Only 9 addresses can be listed.

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.

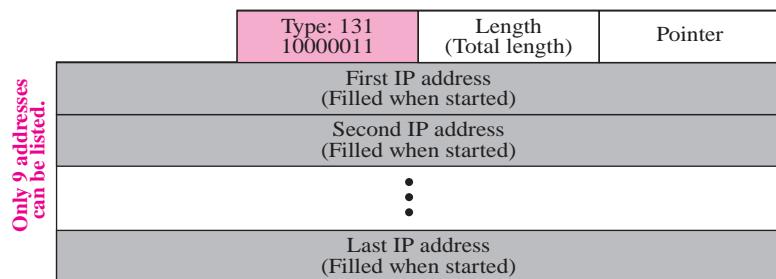
Figure 7.17 Strict-source-route concept



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



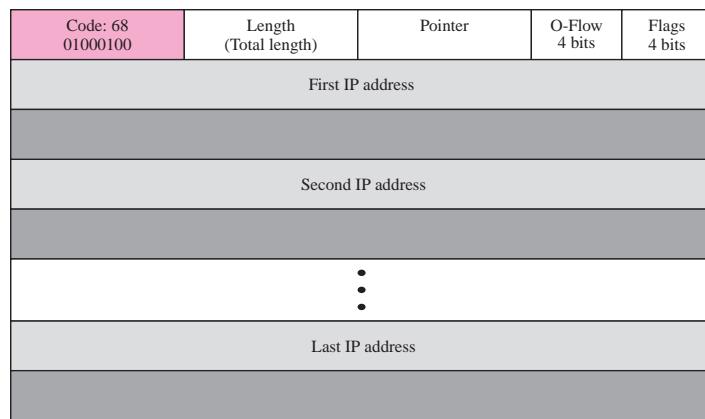
Timestamp

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*

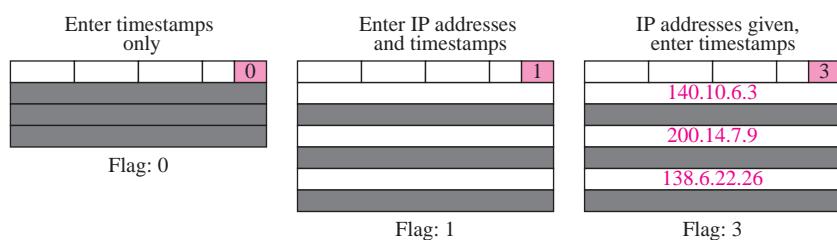
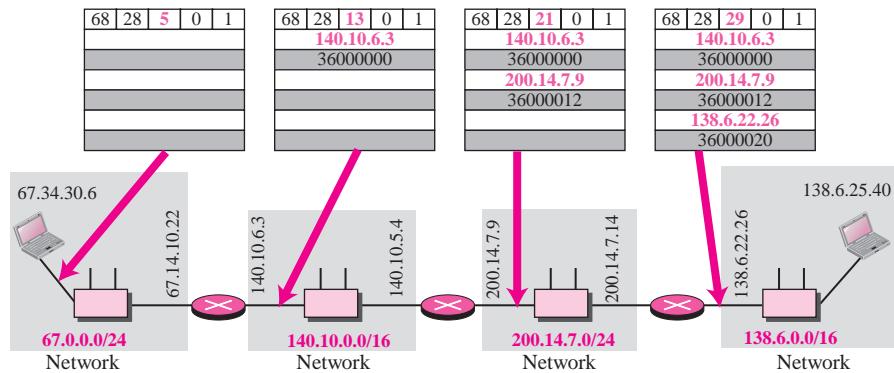


Figure 7.21 shows the actions taken by each router when a datagram travels from source to destination. The figure assumes a flag value of 1.

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 **00000001**; not copied.
 - b. End of option: type is **00000000**; not copied.
 - c. Record route: type is **00000111**; not copied.
 - d. Strict source route: type is **10001001**; copied.
 - e. Loose source route: type is **10000011**; copied.
 - f. Timestamp: type is **01000100**; 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 **00000001**; datagram control.
 - b. End of option: type is **00000000**; datagram control.
 - c. Record route: type is **00000111**; datagram control.
 - d. Strict source route: type is **10001001**; datagram control.
 - e. Loose source route: type is **10000011**; datagram control.
 - f. Timestamp: type is **01000100**; 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:

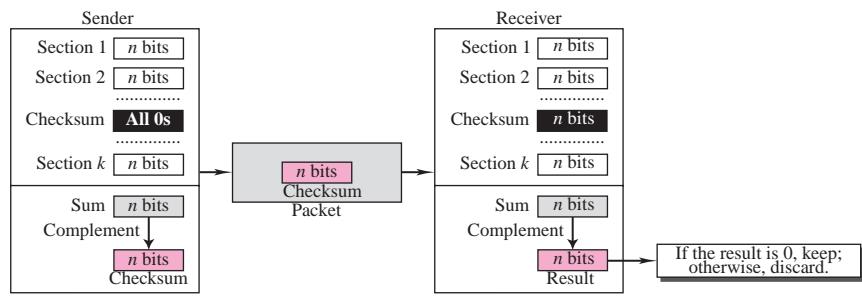
- The packet is divided into k sections, each of n bits.
- All sections are added together using one's complement arithmetic.
- The final result is complemented to make the checksum.

Checksum Calculation at the Receiver

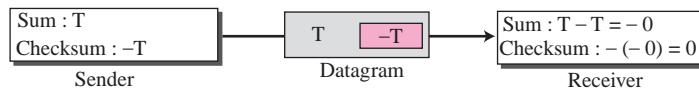
The receiver divides the received packet into k sections and adds all sections. It then complements the result. If the final result is 0, the packet is accepted; otherwise, it is rejected. Figure 7.22 shows graphically what happens at the sender and the receiver.

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

4, 5, and 0	→	01000101	00000000	4	5	0	28
28	→	00000000	00011100		1	0	0
1	→	00000000	00000001			0	
0 and 0	→	00000000	00000000				
4 and 17	→	00000100	00010001				
0	→	00000000	00000000				
10.12	→	00001010	00001100				
14.5	→	00001110	00000101				
12.6	→	00001100	00000110				
7.9	→	00000111	00001001				
Sum	→	01110100	01001110				
Checksum	→	10001011	10110001				

Substitute for 0

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

4, 5, and 0	→	01000101	00000000	4	5	0	28
28	→	00000000	00011100		1	0	0
1	→	00000000	00000001			35761	
0 and 0	→	00000000	00000000				
4 and 17	→	00000100	00010001				
Checksum	→	10001011	10110001				
10.12	→	00001010	00001100				
14.5	→	00001110	00000101				
12.6	→	00001100	00000110				
7.9	→	00000111	00001001				
Sum	→	1111 1111	1111 1111				
Checksum	→	0000 0000	0000 0000				

7.6 IP OVER ATM

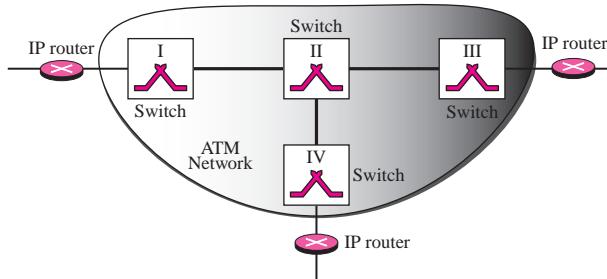
In the previous sections, we assumed that the underlying networks over which the IP datagrams are moving are either LANs or point-to-point WANs. In this section, we want to see how an IP datagram is moving through a switched WAN such as an ATM. We will see that there are similarities as well as differences. 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.

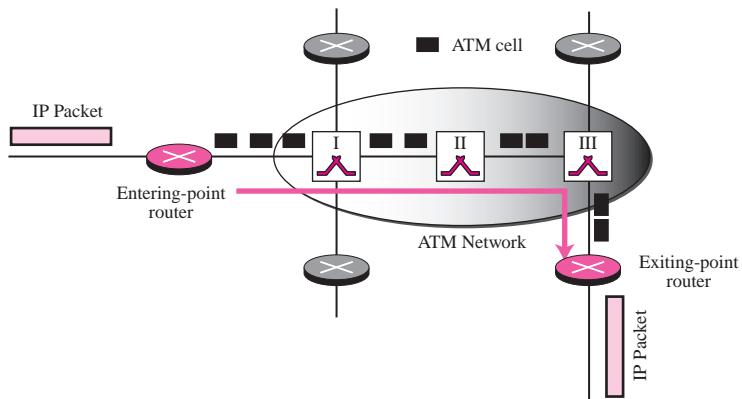
The AAL layer used by the IP protocol is AAL5.

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.

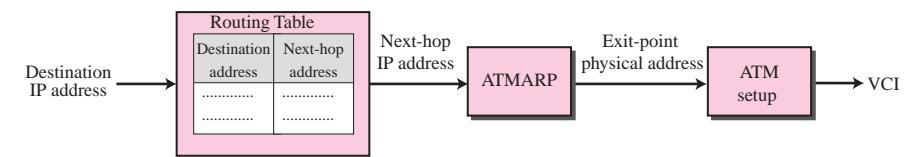
Physical Addresses Each router (or any other device) connected to the ATM network has also a physical address. The physical address is associated with the ATM network and does not have anything to do with the Internet. The ATM Forum defines 20-byte addresses for ATM networks. Each address must be unique 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.

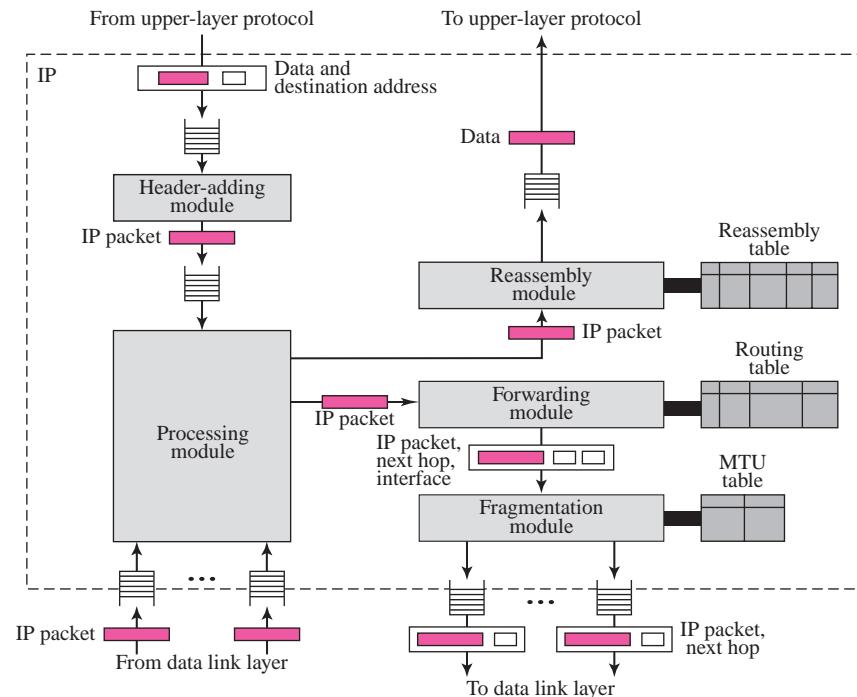
7.8 IP PACKAGE

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
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

```

1  IP_Processing_Module (Datagram)
2  {
3      Remove one datagram from one of the input queues.
4      If (destination address matches a local address)
5      {
6          Send the datagram to the reassembly module.
7          Return.
8      }
9      If (machine is a router)
10     {
11         Decrement TTL.
12     }
13     If (TTL less than or equal to zero)
14     {
15         Discard the datagram.
16         Send an ICMP error message.
17         Return.
18     }
19     Send the datagram to the forwarding module.
20     Return.
21 }
```

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

```

1  IP_Fragmentation_Module (datagram)
2  {
3      Extract the size of datagram
4      If (size > MTU of the corresponding network)
5      {
6          If (D bit is set)
7          {
8              Discard datagram
9              Send an ICMP error message
10             return
11         }
12     Else
13     {
14         Calculate maximum size
15         Divide the segment into fragments
16         Add header to each fragment
17         Add required options to each fragment

```

Table 7.5 Fragmentation module (continued)

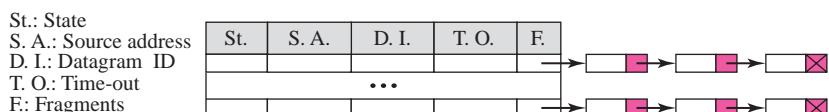
```

18           Send fragment
19           return
20       }
21   }
22 Else
23 {
24   Send the datagram
25 }
26 Return.
27 }

```

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

```

1  IP_Reassembly_Module (datagram)
2  {
3      If (offset value = 0 AND M = 0)
4      {
5          Send datagram to the appropriate queue
6          Return
7      }
8      Search the reassembly table for the entry
9      If (entry not found)
10     {
11         Create a new entry
12     }
13     Insert datagram into the linked list
14     If (all fragments have arrived)
15     {
16         Reassemble the fragment
17         Deliver the fragment to upper-layer protocol
18         return
19     }
20     Else
21     {
22         If (time-out expired)
23         {
24             Discard all fragments
25             Send an ICMP error message
26         }
27     }
28     Return.
29 }
```

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	maximum transfer unit (MTU)
checksum	no-operation option
codepoint	output queue
datagram	ping
destination address	pointer field
differentiated services	precedence
end-of-option option	processing module
entering-point router	reassemble module
existing-point router	reassemble table
forwarding module	record-route option
fragmentation	service type
fragmentation module	source address
fragmentation offset	strict-source-route option
header length	time to live
header-adding module	timestamp option
input queue	traceroute
Internet Protocol (IP)	type field
length field	type of service (TOS)
loose-source-route option	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.

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.

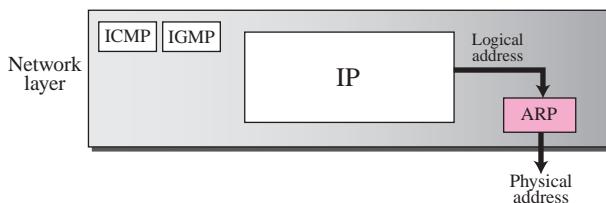
Dynamic Mapping

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

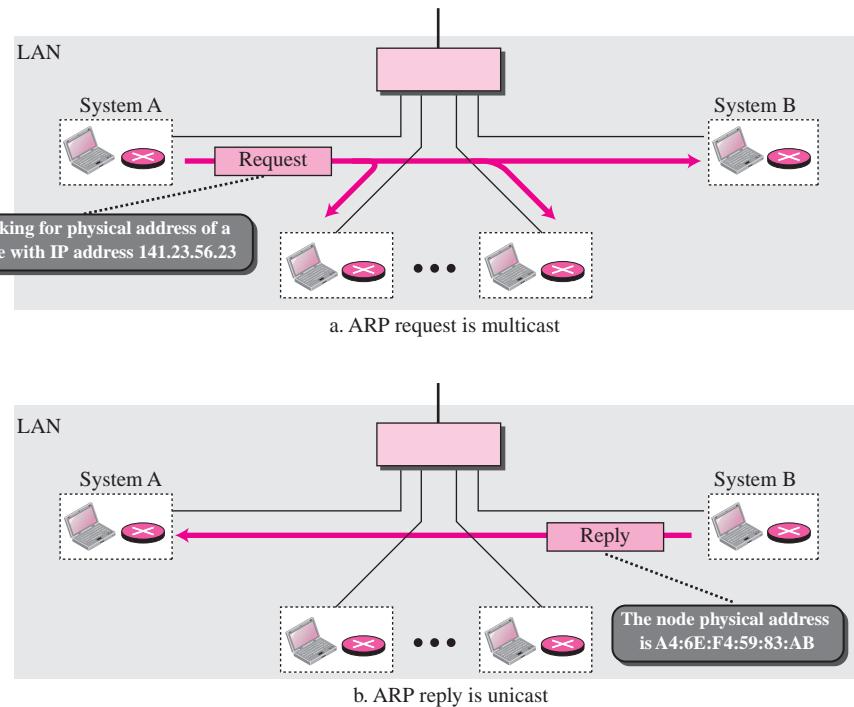


ARP associates an IP address with its physical address. On a typical physical network, such as a LAN, each device on a link is identified by a physical or station address that is usually imprinted on the NIC.

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

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

Hardware Type		Protocol Type
Hardware length	Protocol length	Operation Request 1, Reply 2
Sender hardware address (For example, 6 bytes for Ethernet)		
Sender protocol address (For example, 4 bytes for IP)		
Target hardware address (For example, 6 bytes for Ethernet) (It is not filled in a request)		
Target protocol address (For example, 4 bytes for IP)		

- ❑ **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.

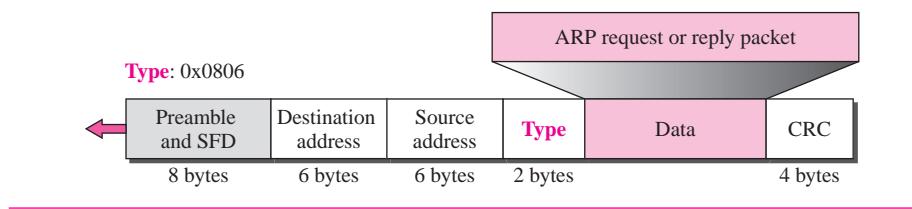
Operation

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.

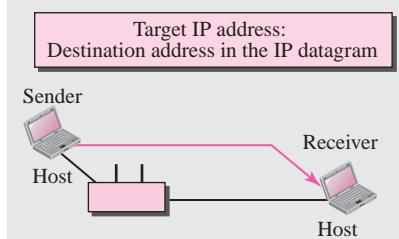
Four Different Cases

The following are four different cases in which the services of ARP can be used (see Figure 8.5).

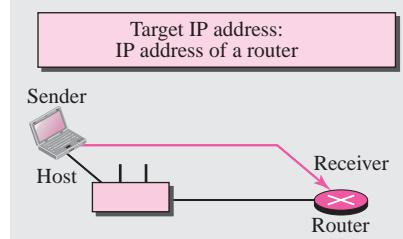
- ❑ **Case 1:** The sender is a host and wants to send a packet to another host on the same network. In this case, the logical address that must be mapped to a physical address is the destination IP address in the datagram header.
- ❑ **Case 2:** The sender is a host and wants to send a packet to another 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

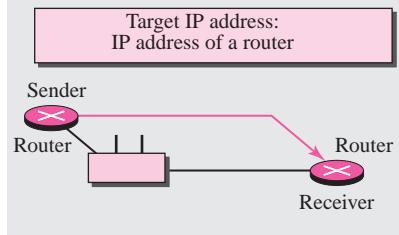
Case 1: A host has a packet to send to a host on the same network.



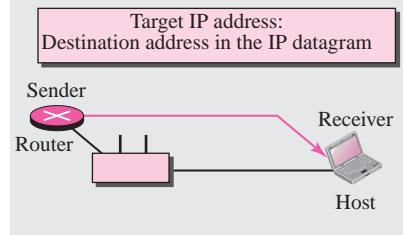
Case 2: A host has a packet to send to a host on another network.



Case 3: A router has a packet to send to a host on another network.



Case 4: A router has a packet to send to a host on the same network.



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.

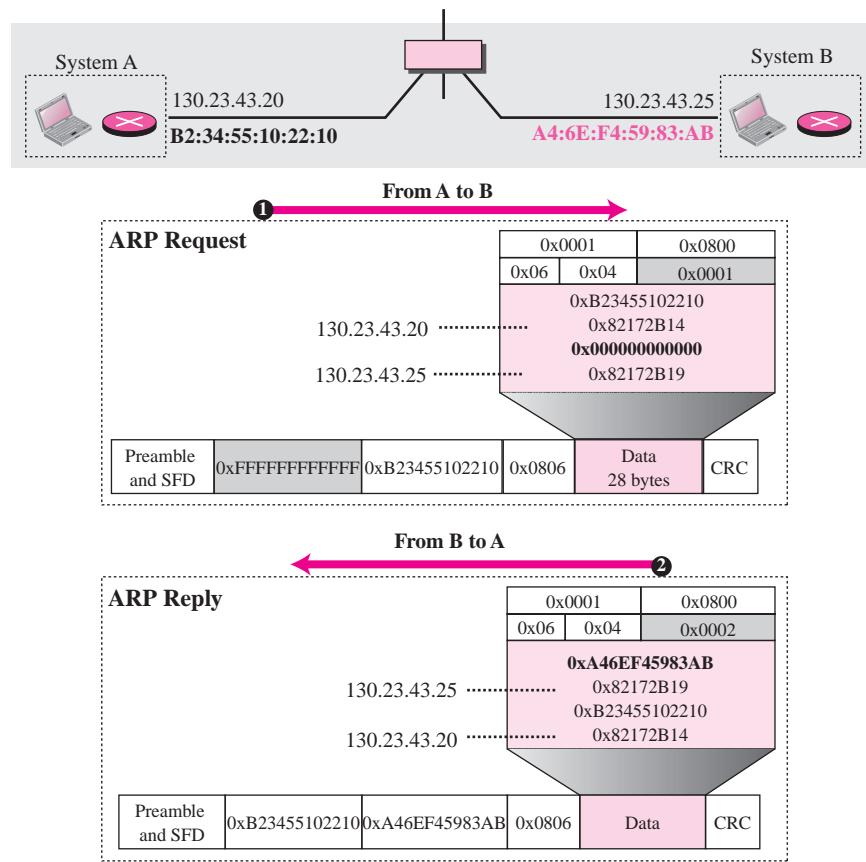
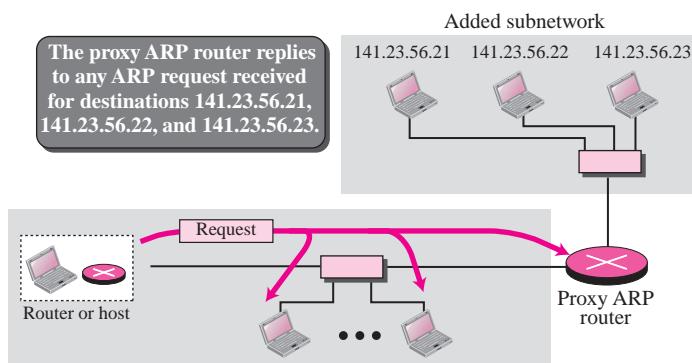
Solution

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 packets 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

Hardware Type		Protocol Type	
Sender Hardware Length	Reserved	Operation	
Sender Protocol Length	Target Hardware Length	Reserved	Target Protocol Length
Sender hardware address (20 bytes)			
Sender protocol address			
Target hardware address (20 bytes)			
Target protocol address			

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

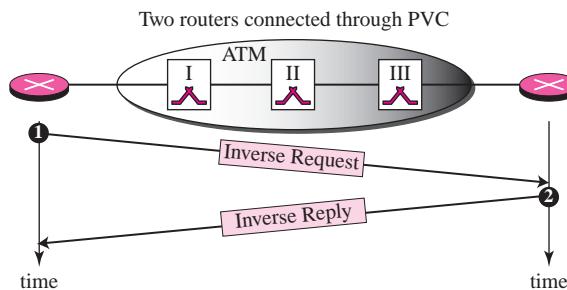
A permanent virtual circuit (PVC) connection is established between two end points by the network provider. The VPIs and VCIs are defined for the permanent connections and the values are entered in a table for each switch.

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

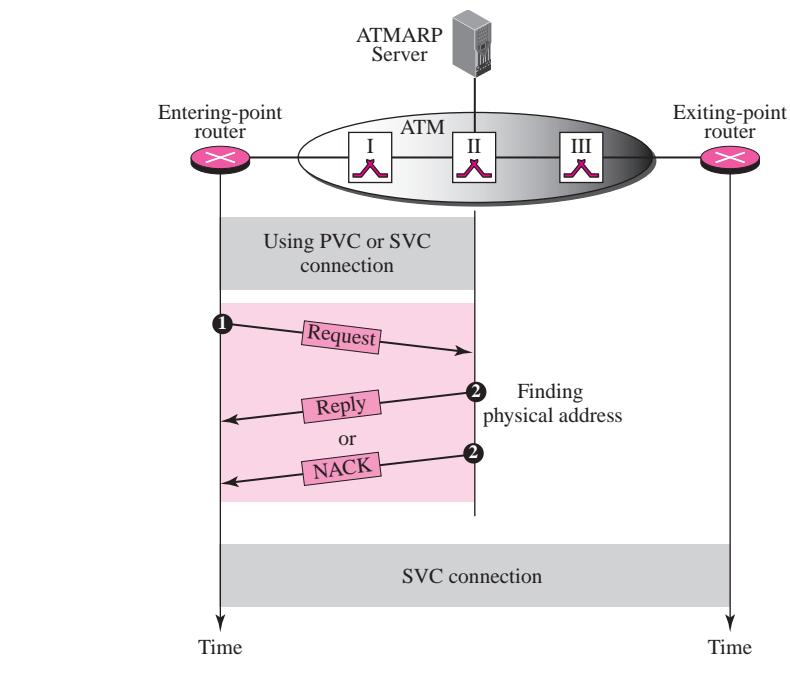


SVC Connection

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.

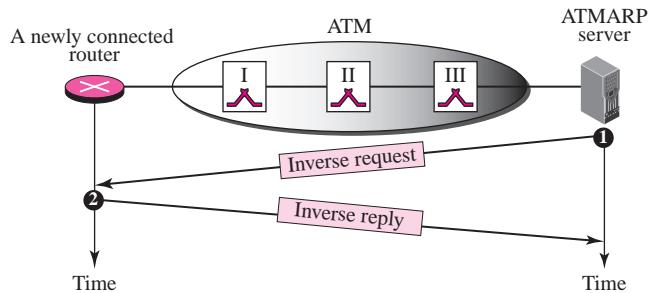
The request and reply message can be used to bind a physical address to an IP address in an SVC situation.

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



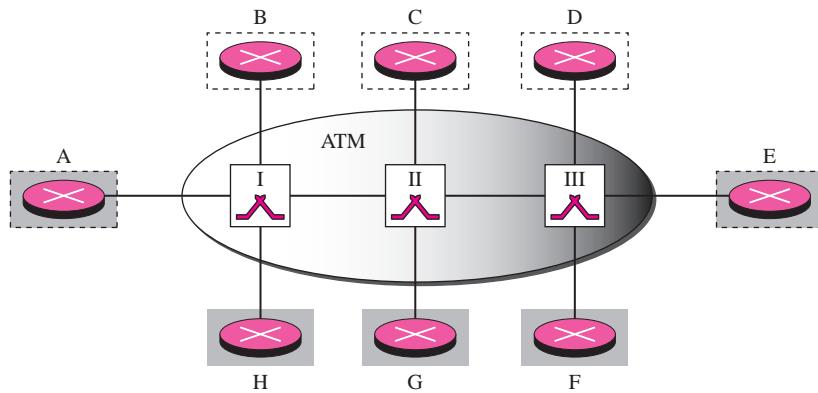
Logical IP Subnet (LIS)

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.

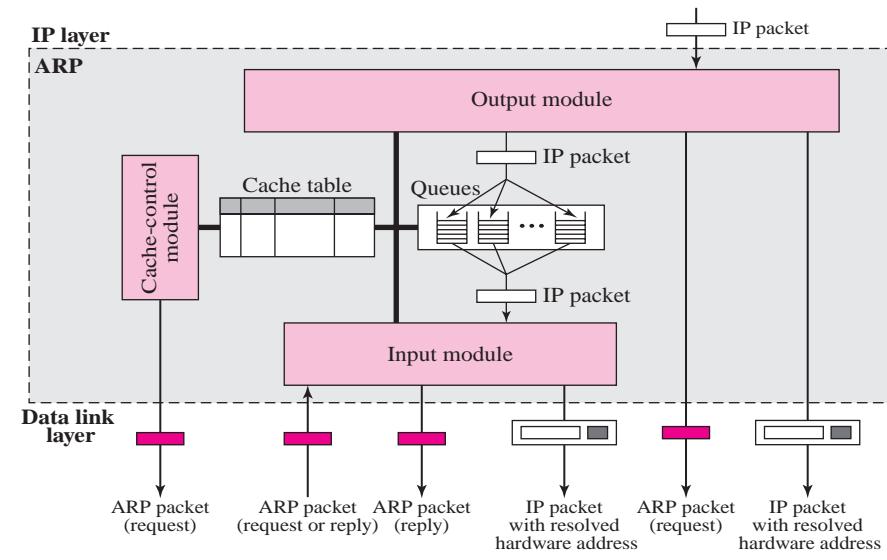
We can say that this ARP package involves five components: a **cache table**, queues, an output module, an input module, and a cache-control module. The package receives an IP datagram that needs to be encapsulated in a frame that needs the destination physical (hardware) address. If the ARP package 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*

```
1  ARP_Output_Module ( )
2  {
3      Sleep until an IP packet is received from IP software.
4      Check cache table for an entry corresponding to the
5          destination of IP packet.
6      If (entry is found)
7      {
8          If (the state is RESOLVED)
9          {
10             Extract the value of the hardware address from the entry.
11             Send the packet and the hardware address to data
12                 link layer.
13             Return
14         } // end if
15         If (the state is PENDING)
16         {
17             Enqueue the packet to the corresponding queue.
18             Return
19         } // end if
20     } // end if
21     If (entry is not found)
22     {
23         Create a cache entry with state set to PENDING and
24             ATTEMPTS set to 1.
25         Create a queue.
26         Enqueue the packet.
27         Send an ARP request.
28         Return
29     } // end if
30 } // 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

```
1 ARP_Cache_Control_Module ( )
2 {
3     Sleep until the periodic timer matures.
4     Repeat for every entry in the cache table
5     {
6         If (the state is FREE)
7         {
8             Continue.
9         } //end if
10        If (the state is PENDING)
11        {
```

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)	logical IP subnet (LIS)
cache table	physical address
cache-control module	proxy ARP
dynamic mapping	queue
encapsulation	reserved field
inverse reply message	Reverse Address Resolution Protocol
inverse request message	(RARP)
IP addresses	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?

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.

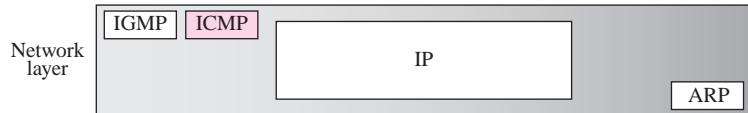
9.1 INTRODUCTION

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.

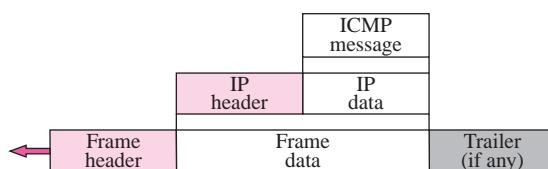
The **Internet Control Message Protocol (ICMP)** has been designed to compensate for the above two deficiencies. It is a companion to the IP protocol. Figure 9.1 shows the position of ICMP in relation to IP and other protocols in the network layer.

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

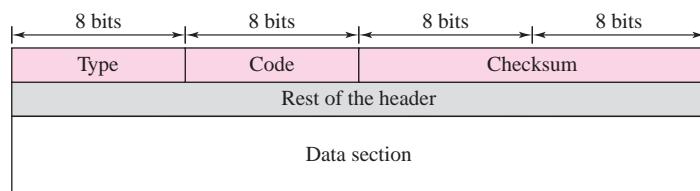
Category	Type	Message
Error-reporting messages	3	Destination unreachable
	4	Source quench
	11	Time exceeded
	12	Parameter problem
	5	Redirection
Query messages	8 or 0	Echo request or reply
	13 or 14	Timestamp request or reply

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

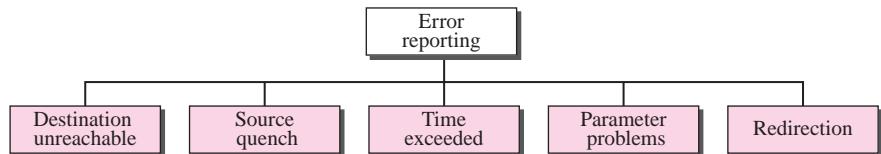
One of the main responsibilities of ICMP is to report errors. Although technology has produced increasingly reliable transmission media, errors still exist and must be handled. IP, as discussed in Chapter 7, is an unreliable protocol. This means that error

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



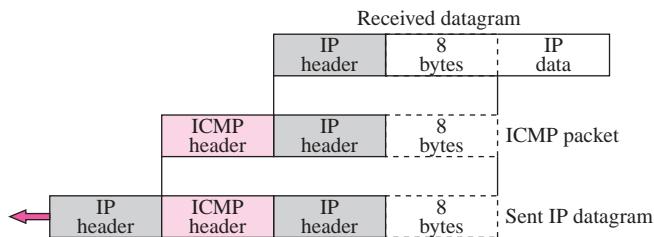
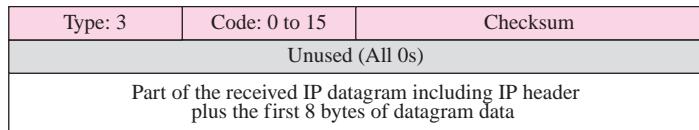
The following are important points about ICMP error messages:

- ❑ 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

Type: 4	Code: 0	Checksum
Unused (All 0s)		
Part of the received IP datagram including IP header plus the first 8 bytes of datagram data		

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

Type: 11	Code: 0 or 1	Checksum
Unused (All 0s)		
Part of the received IP datagram including IP header plus the first 8 bytes of datagram data		

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.

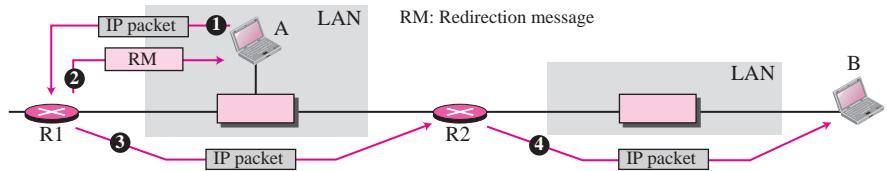
Figure 9.9 Parameter-problem message format

Type: 12	Code: 0 or 1	Checksum
Pointer	Unused (All 0s)	
Part of the received IP datagram including IP header plus the first 8 bytes of datagram data		

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

Type: 5	Code: 0 to 3	Checksum
IP address of the target router		
Part of the received IP datagram including IP header plus the first 8 bytes of datagram data		

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

Type 8: Echo request	Type: 8 or 0	Code: 0	Checksum
Type 0: Echo reply	Identifier	Sequence number	
Optional data Sent by the request message; repeated by the reply message			

Figure 9.13 Timestamp-request and timestamp-reply message format

Type 13: request	Type: 13 or 14	Code: 0	Checksum
Type 14: reply	Identifier	Sequence number	
Original timestamp			
Receive timestamp			
Transmit timestamp			

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:

- Information request and replay* messages are not used today because their duties are done by Address Resolution Protocol (ARP) discussed in Chapter 8.
- 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.
- Router solicitation and advertisement* 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

8	0	0	
1		9	
TEST			
8 & 0	00001000	00000000	
0	00000000	00000000	
1	00000000	00000001	
9	00000000	00001001	
T & E	01010100	01000101	
S & T	01010011	01010100	
Sum	10101111	10100011	
Checksum	01010000	01011100	

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.

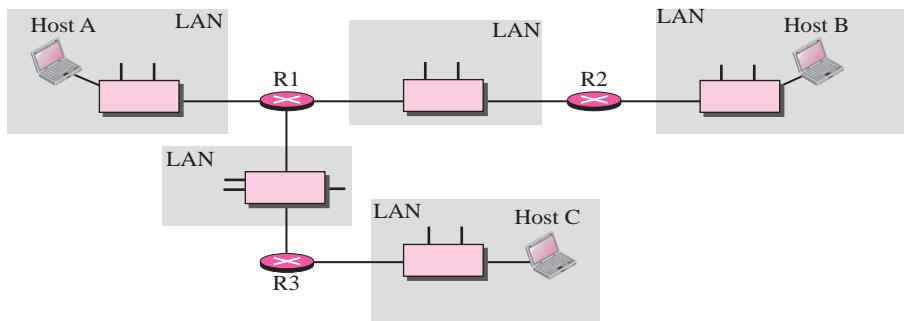
```
$ ping mail.adelphia.net
PING mail.adelphia.net (68.168.78.100) 56(84) bytes of data.
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=0 ttl=48 time=85.4 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=1 ttl=48 time=84.6 ms
64 bytes from mail.adelphia.net (68.168.78.100): icmp_seq=2 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
2	Ddmz.fhda.edu	(153.18.251.40)	2.132 ms	2.266 ms
3	Cinic.fhda.edu	(153.18.253.126)	2.110 ms	2.145 ms
4	cenic.net	(137.164.32.140)	3.069 ms	2.875 ms
5	cenic.net	(137.164.22.31)	4.205 ms	4.870 ms
6	cenic.net	(137.164.22.167)	4.250 ms	4.159 ms
7	cogentco.com	(38.112.6.225)	5.062 ms	4.825 ms
8	cogentco.com	(66.28.4.69)	6.070 ms	6.207 ms
9	cogentco.com	(66.28.4.94)	6.070 ms	5.928 ms
10	cogentco.com	(154.54.2.226)	6.545 ms	6.399 ms
11	sbcglo	(151.164.89.241)	6.379 ms	6.370 ms
12	sbcglo	(64.161.1.45)	6.908 ms	6.748 ms
13	sbcglo	(64.161.1.29)	7.023 ms	7.040 ms
14	snfc21.pbi.net	(151.164.191.49)	7.656 ms	7.129 ms
15	sbcglobal.net	(151.164.243.58)	7.844 ms	7.545 ms
16	pacbell.net	(209.232.138.114)	9.857 ms	9.535 ms
17	209.233.48.223	(209.233.48.223)	10.634 ms	10.771 ms
18	alpha.Xerox.COM	(13.1.64.93)	10.922 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

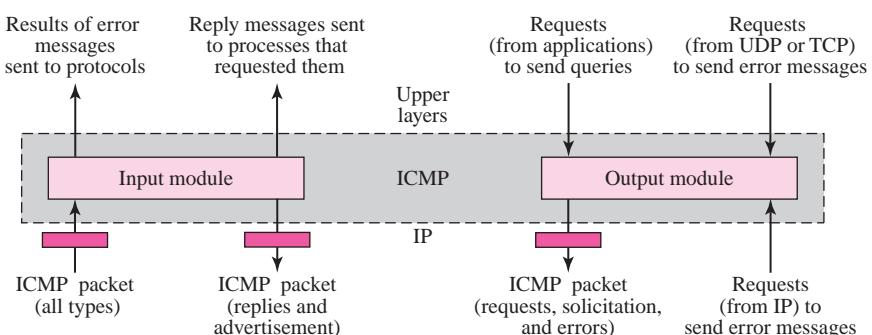
Finally, we use the traceroute program to find the route between fhda.edu and mhhe.com (McGraw-Hill server). We notice that we cannot find the whole route. When traceroute does not receive a response within 5 seconds, it prints an asterisk to signify a problem (not the case in this example), and then 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*

```

1  ICMP_Input_Module (ICMP_Packet)
2  {
3      If (the type is a request)
4      {
5          Create a reply
6          Send the reply
7      }
8      If (the type defines a redirection)
9      {
10         Modify the routing table
11     }
12     If (the type defines other error messages)
13     {
14         Inform the appropriate source protocol
15     }
16     Return
17 }
```

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*

```

1  ICMP_Output_Module (demand)
2  {
3      If (the demand defines an error message)
4      {
```

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	redirection message
echo-reply messages	round-trip time (RTT)
error-reporting message	source-quench message
echo-request message	time-exceeded message
Internet Control Message Protocol (ICMP)	timestamp-reply message
parameter-problem message	timestamp-request message
ping	traceroute
query message	

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.

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

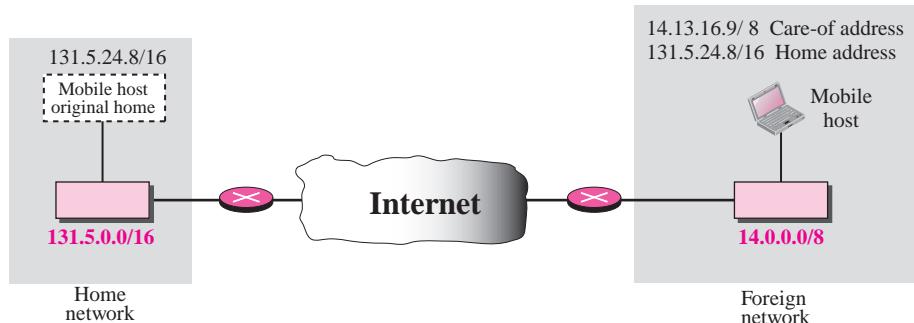
One simple solution is to let the **mobile host** change its address as it goes to the new network. The host can use DHCP (see Chapter 18) to obtain a new address to associate it with the new network. This approach has several drawbacks. First, the configuration files would need to be changed. Second, 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



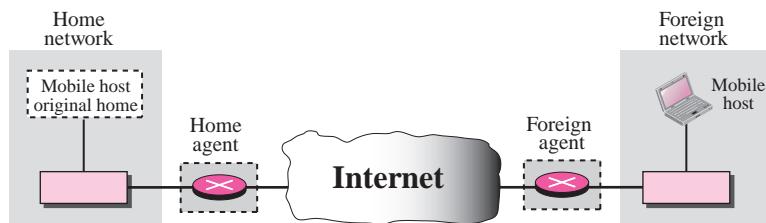
Mobile IP has two addresses for a mobile host: one home address and one care-of address. The home address is permanent; the care-of address changes as the mobile host moves from one network to another.

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.

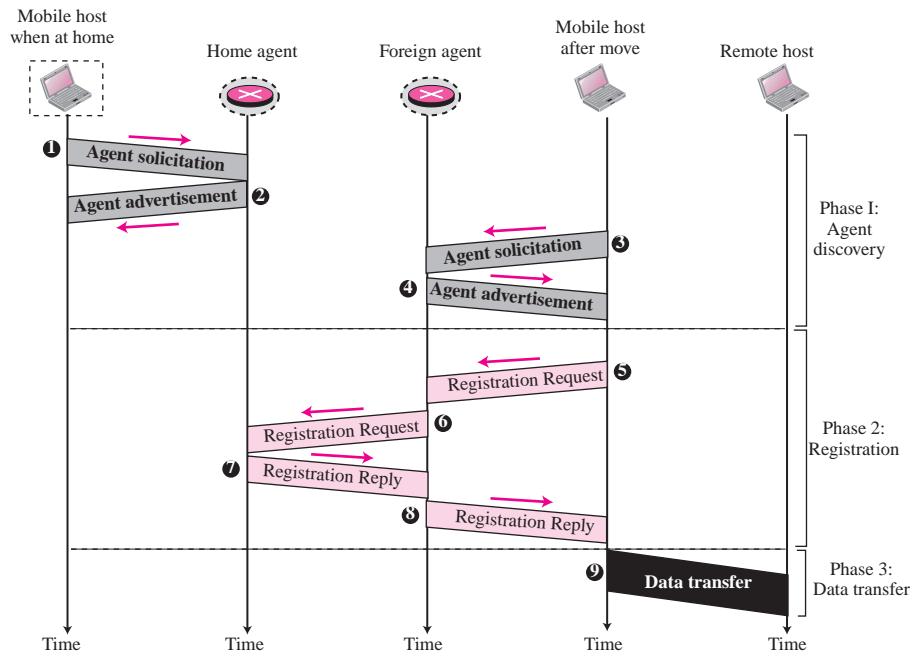
10.3 THREE PHASES

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

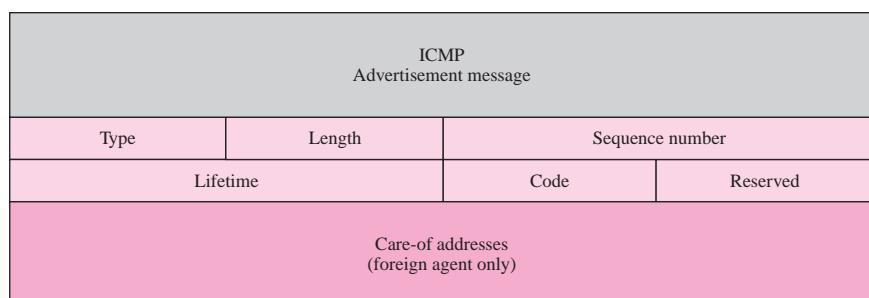
The first phase in mobile communication, **agent discovery**, consists of two subphases. A mobile host must discover (learn the address of) a home agent before it leaves its home network. A mobile host must also discover a foreign agent after it has moved to a foreign network. This discovery consists 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

Type	Flag	Lifetime
		Home address
		Home agent address
		Care-of address
		Identification
		Extensions ...

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

Type	Code	Lifetime
	Home address	
	Home agent address	
	Identification	
	Extensions ...	

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.

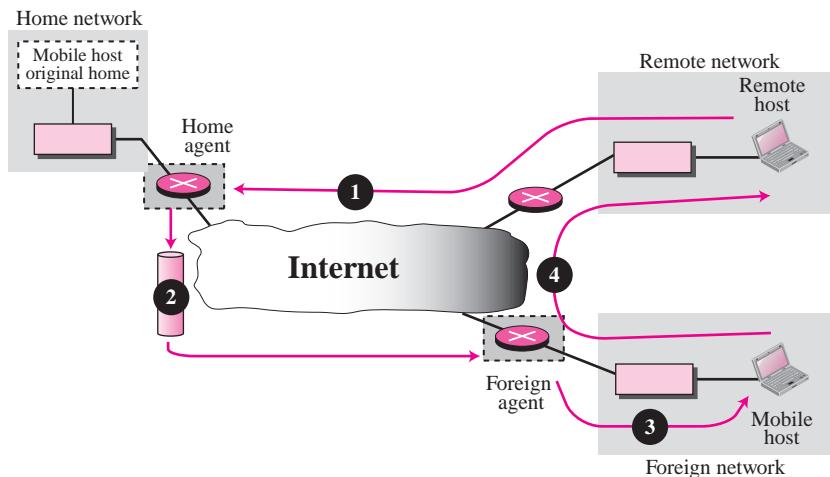
A registration request or reply is sent by UDP using the well-known port 434.

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.

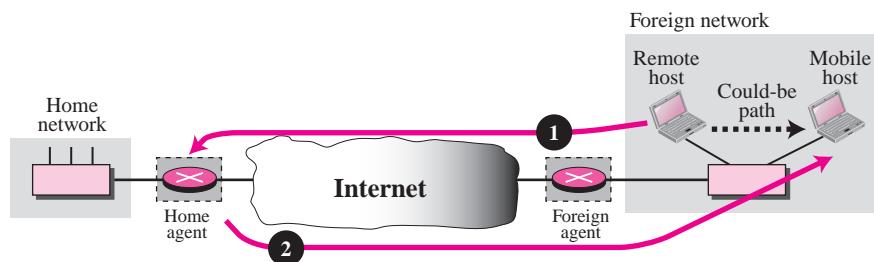
10.4 INEFFICIENCY IN MOBILE IP

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.

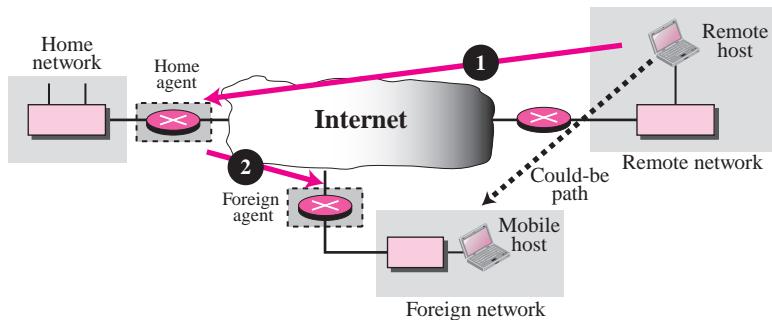
Since a computer usually communicates with other local computers (principle of locality), the inefficiency from double crossing is significant.

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	triangle routing
registration reply	update binding packet
registration request	warning packet
stationary host	

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.

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

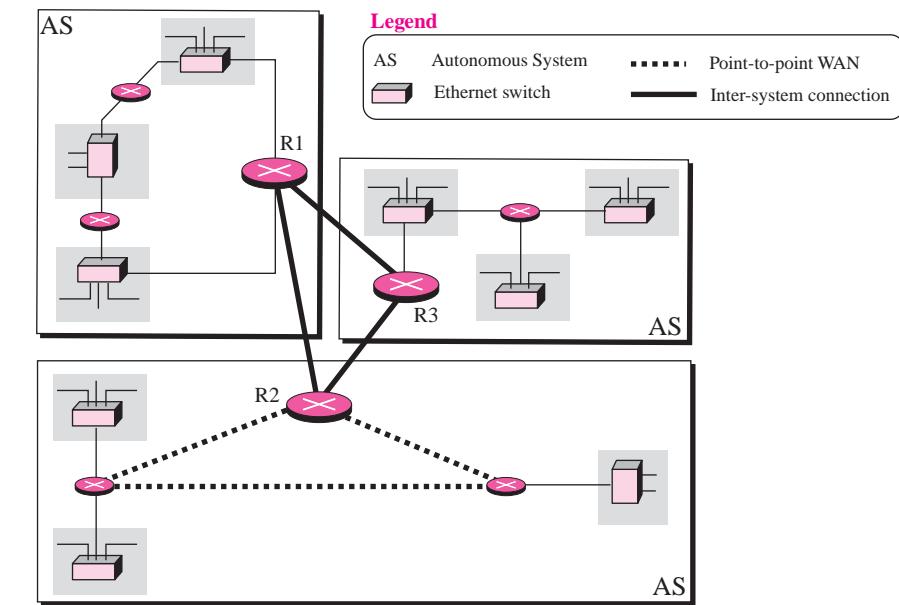
Routing protocols have been created in response to the demand for dynamic routing tables. A routing protocol is a combination of rules and procedures that lets routers in the internet inform each other of changes. It allows routers to share whatever they know about the internet or their neighborhood. 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 *interdomain routing*. We start the next section with defining these terms.

11.2 INTRA- AND INTER-DOMAIN ROUTING

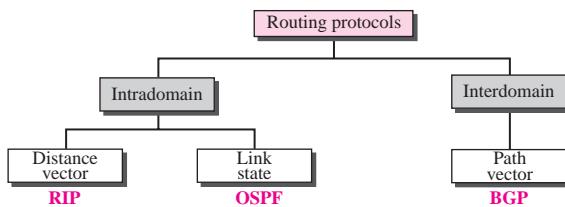
Today, an internet can be so large that one routing protocol cannot handle the task of updating the routing tables of all routers. For this reason, an internet is divided into autonomous systems. An **autonomous system (AS)** is a group of networks and routers under the authority of a single administration. 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.

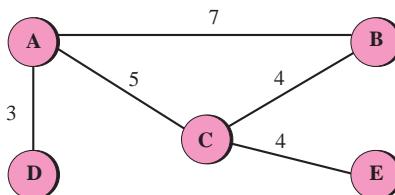
Figure 11.2 Popular routing protocols

11.3 DISTANCE VECTOR ROUTING

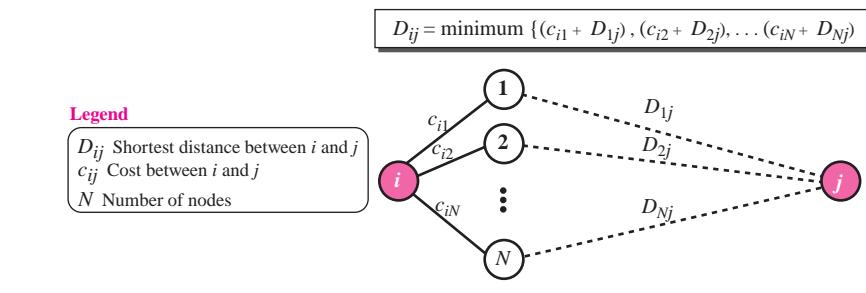
We first discuss **distance vector routing**. This method sees an AS, with all routers and networks, as a *graph*, a set of nodes and lines (edges) connecting the nodes. A router can normally be represented by a node and a network by a link connecting two nodes, although other representations are also 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 Table $_i$, to the three columns in row i as Table $_i$.dest, Table $_i$.cost, and 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          Tablei.dest = address of the attached network
7          Tablei.cost = 1
8          Tablei.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 < Ti.cost      or      R.next == Ti.next)
30         {
31             Ti.cost = R.cost + 1
32             Ti.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         TN +1.dest = R.dest
40         TN +1.cost = R.cost + 1
41         TN +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.

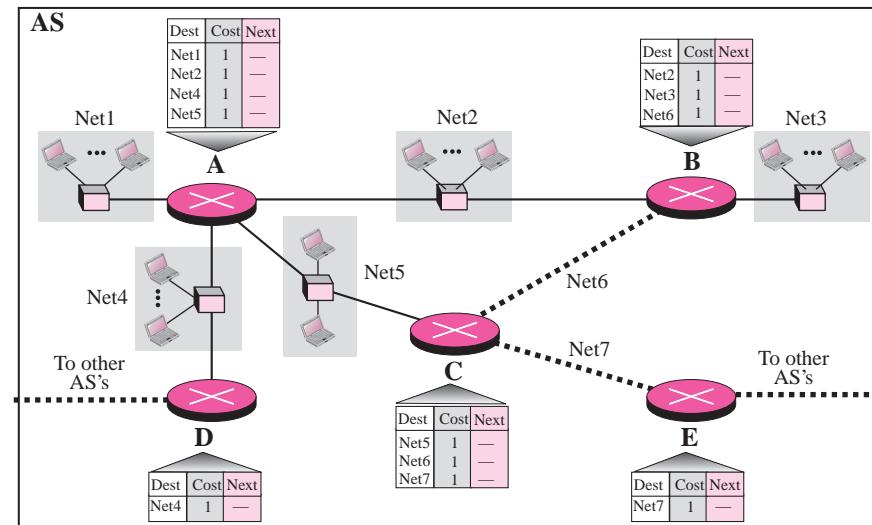
Lines 23 to 47 give the details of updating process. When a record arrives, the router searches for the destination address in the routing table.

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

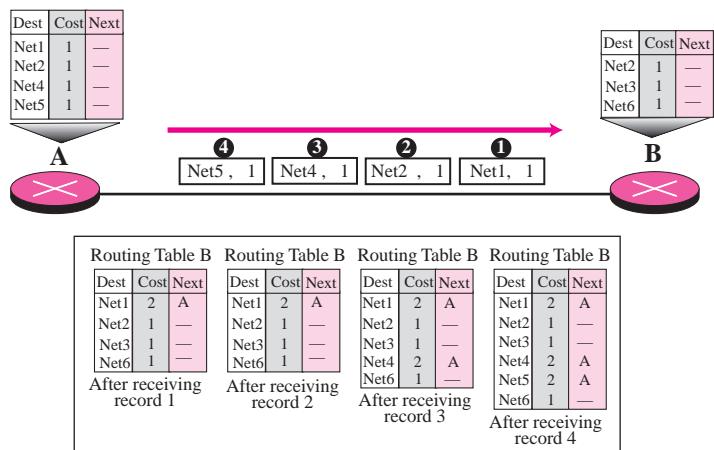
Figure 11.5 shows the initial routing table for an AS. Note that the figure does not mean that all routing tables have been created at the same time; each router creates its own routing table when it is booted.

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

- 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.
- 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.
- When router B receives record 3, it searches its router, and since Net4 is not found, it is added to the table.
- 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

A			B			C			D			E		
Dest	Cost	Next												
Net1	1	—	Net1	2	A	Net1	2	A	Net1	2	A	Net1	3	C
Net2	1	—	Net2	1	—	Net2	2	A	Net2	2	A	Net2	3	C
Net3	2	B	Net3	1	—	Net3	2	B	Net3	3	A	Net3	3	C
Net4	1	—	Net4	2	A	Net4	2	A	Net4	1	—	Net4	3	C
Net5	1	—	Net5	2	A	Net5	1	—	Net5	1	A	Net5	2	C
Net6	2	C	Net6	1	—	Net6	1	—	Net6	3	A	Net6	2	C
Net7	2	C	Net7	2	C	Net7	1	—	Net7	3	A	Net7	1	—

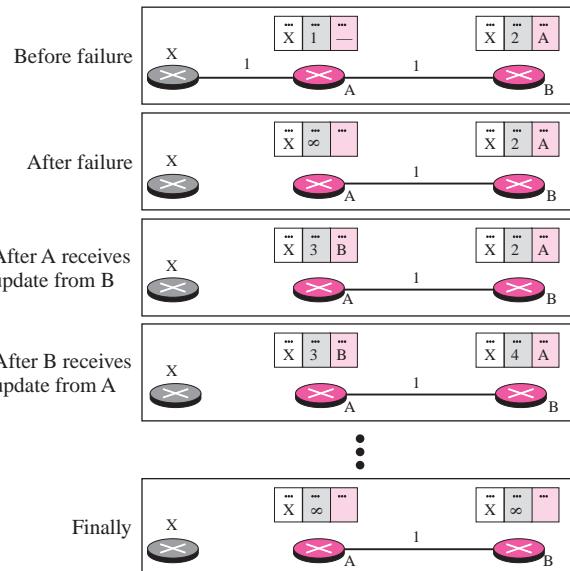
Count to Infinity

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.

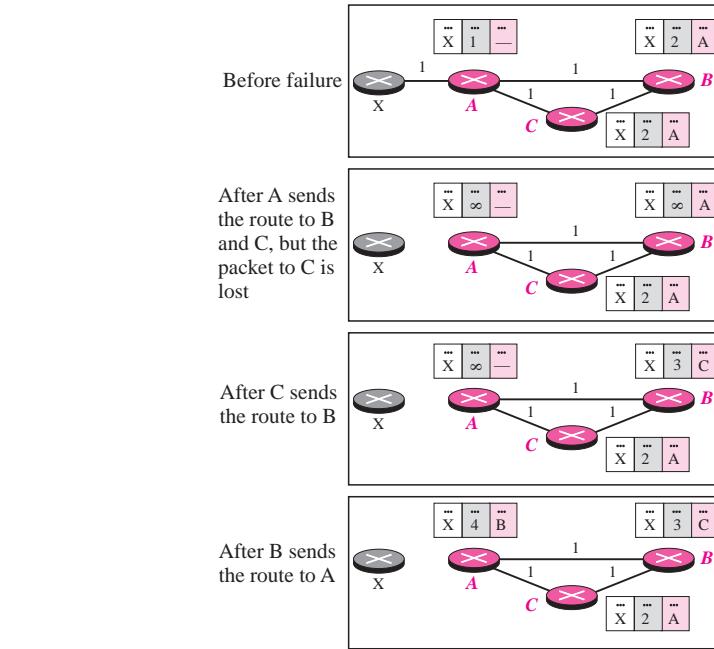
Split Horizon Another solution is called **split horizon**. In this strategy, instead of flooding the table through each interface, each node sends only part of its table through each interface. If, according to its table, node B thinks that the optimum route to reach X is via A, it does not need to 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

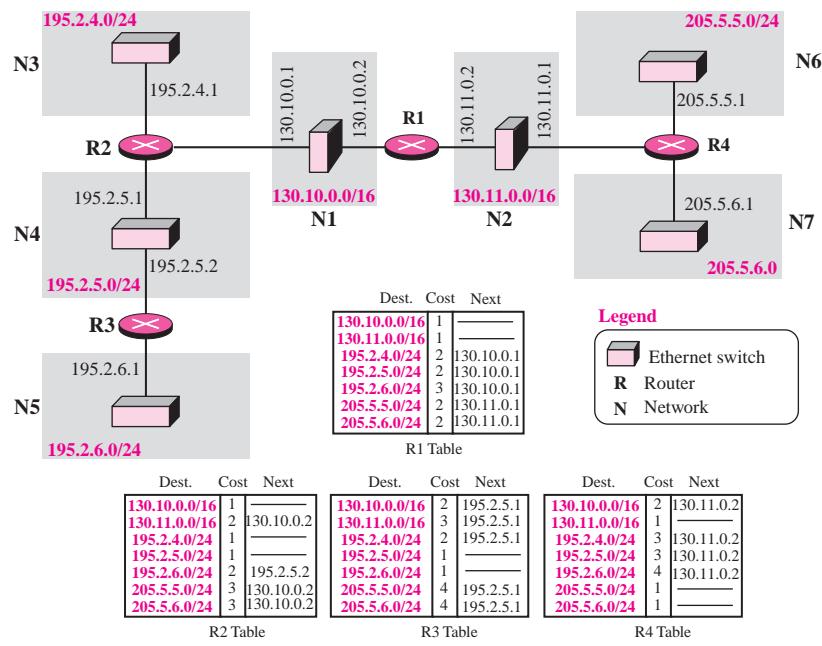
The **Routing Information Protocol (RIP)** is an intradomain (interior) routing protocol used inside an autonomous system. It is a very simple protocol based on distance vector routing. RIP implements distance vector routing directly with some considerations:

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.

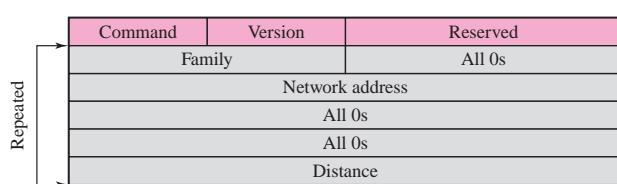
Figure 11.10 Example of a domain using RIP



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

a. Request for some

b. Request for all

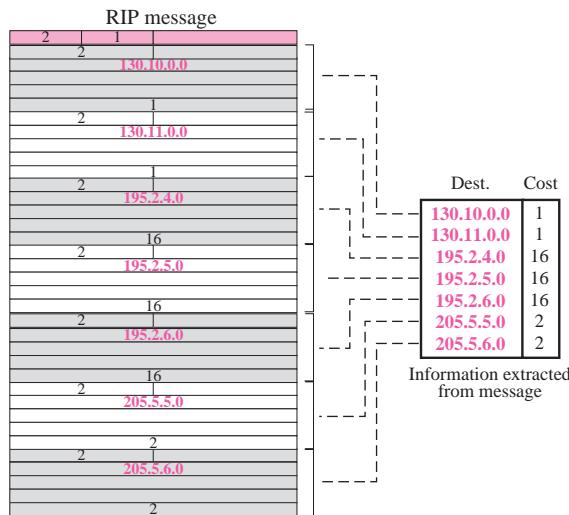
Response

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.

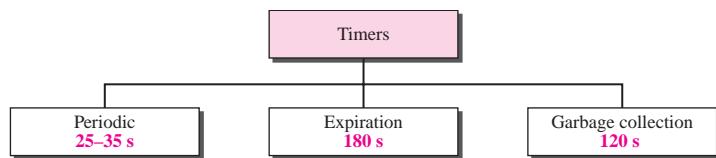
The message is prepared with the combination of split horizon and poison reverse strategy in mind. Router R1 has obtained information about networks 195.2.4.0, 195.2.5.0, and 195.2.6.0 from router R2. When R1 sends an update message to R2, it replaces the actual value of the hop counts for these three 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

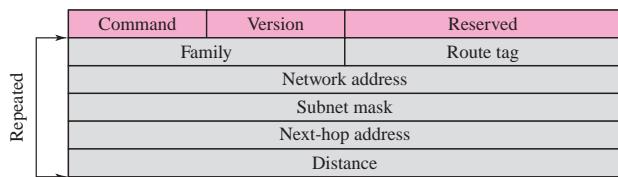
RIP Version 2

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.

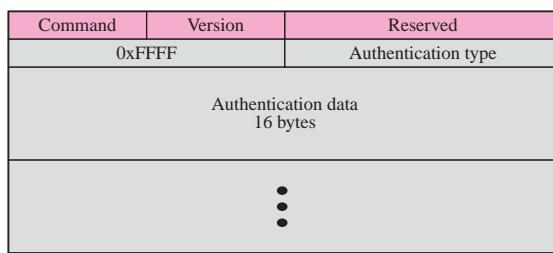
Figure 11.15 RIP version 2 format

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

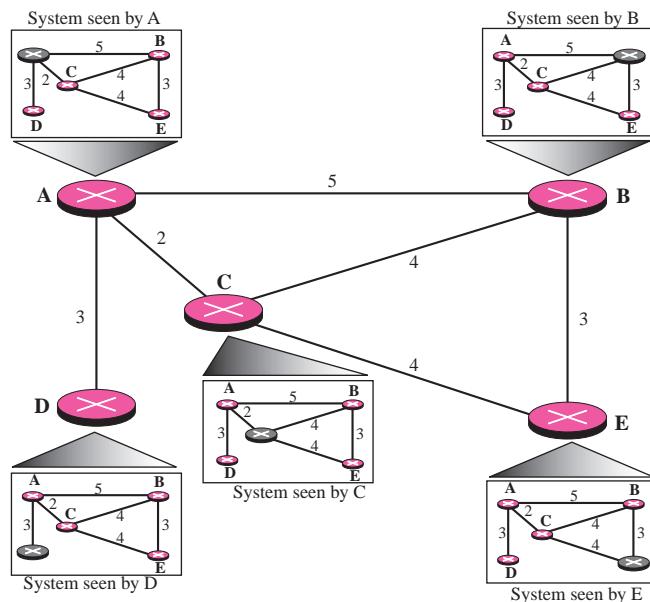
RIP messages are encapsulated in UDP user datagrams. A RIP message does not include a field that indicates the length of the message. This can be determined from the UDP packet. The well-known port assigned to RIP in UDP is port 520.

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

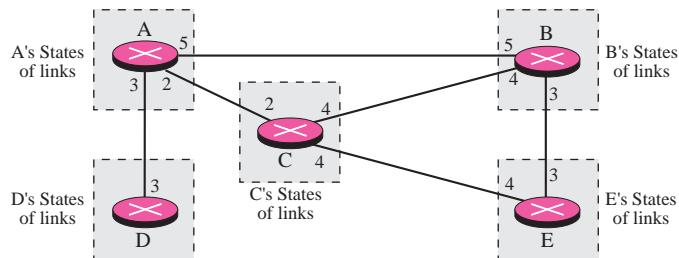


The figure shows a simple domain with five nodes. Each node uses the same topology to create a routing table, but the routing table for each node is unique because the calculations are based on different interpretations of the topology. This is analogous to a city map. Two persons in two different cities 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.

$$D_j = \min(D_p, D_i + c_{ij}) \quad \text{for all remaining nodes}$$

Table 11.3 shows the simple version of this algorithm.

Table 11.3 *Dijkstra's Algorithm*

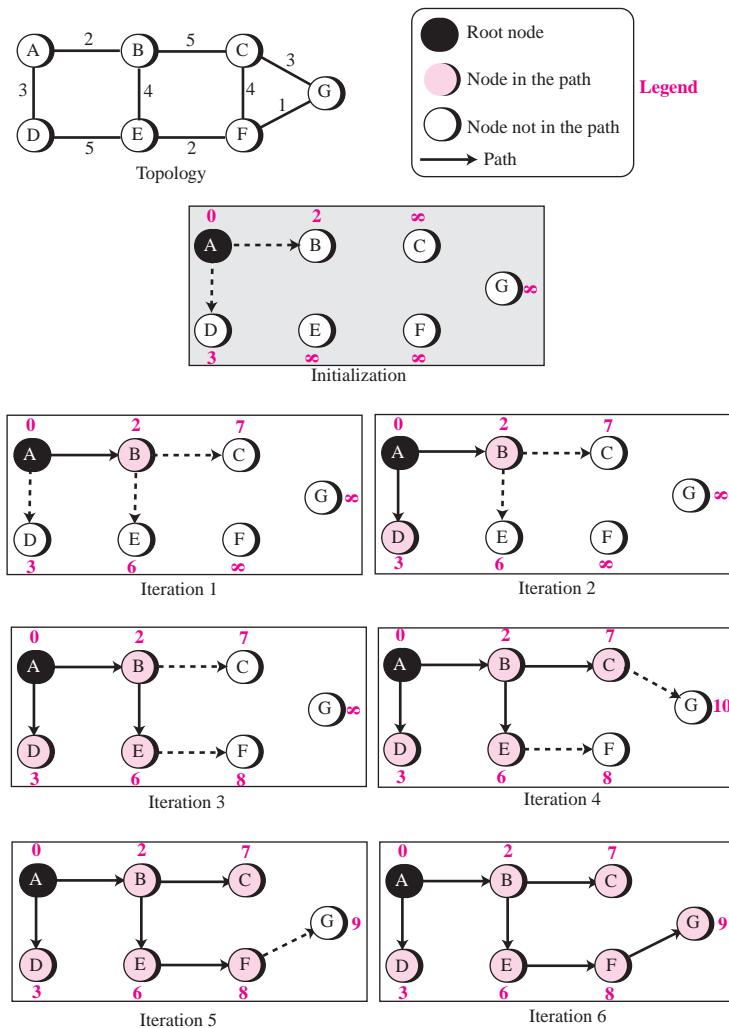
```

1 Dijkstra ( )
2 {
3     // Initialization
4     Path = {s}           // s means self
5     for (i = 1 to N)
6     {
7         if (i is a neighbor of s and i ≠ s)   Di = csi
8         if (i is not a neighbor of s)          Di = ∞
9     }
10    Ds = 0
11
12 } // Dijkstra
13 // Iteration
14 Repeat
15 {
16     // Finding the next node to be added
17     Path = Path ∪ i   if Di is minimum among all remaining nodes
18
19     // Update the shortest distance for the rest
20     for (j = 1 to M) // M number of remaining nodes
21     {
22         Dj = minimum (Dj ,  Dj + cij)
23     }
24 } until (all nodes included in the path, M = 0)
25

```

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.

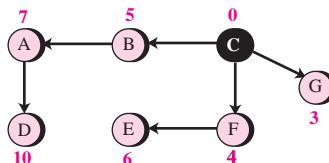
Figure 11.19 Forming shortest path tree for router A in a graph

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.

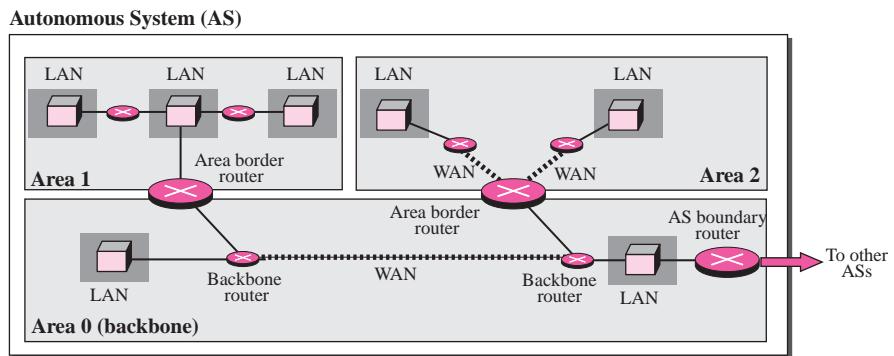
Routers inside an area flood the area with routing information. At the border of an area, special routers called **area border routers** summarize the information about the area and send it to other areas. Among the areas inside an autonomous system is a special area called the *backbone*; all of the 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



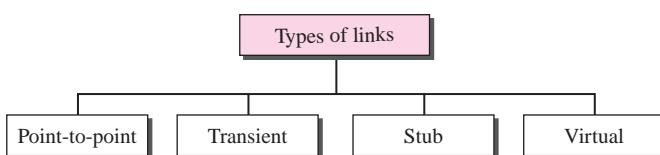
Metric

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).

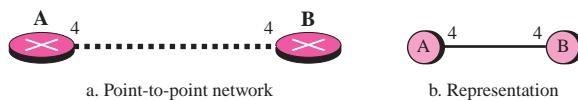
Figure 11.22 Types of links



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).

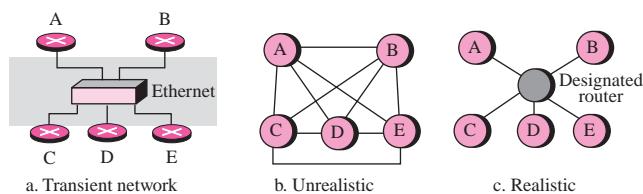
Figure 11.23 Point-to-point link



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



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.

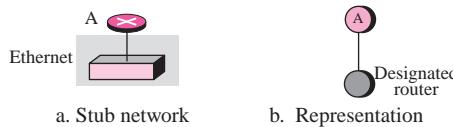
To show that each router is connected to every other router through one single network, the network itself is represented by a node. However, because a network is not a machine, it cannot function as a router. One of the routers in the network takes this responsibility. It is assigned a dual purpose; 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



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.

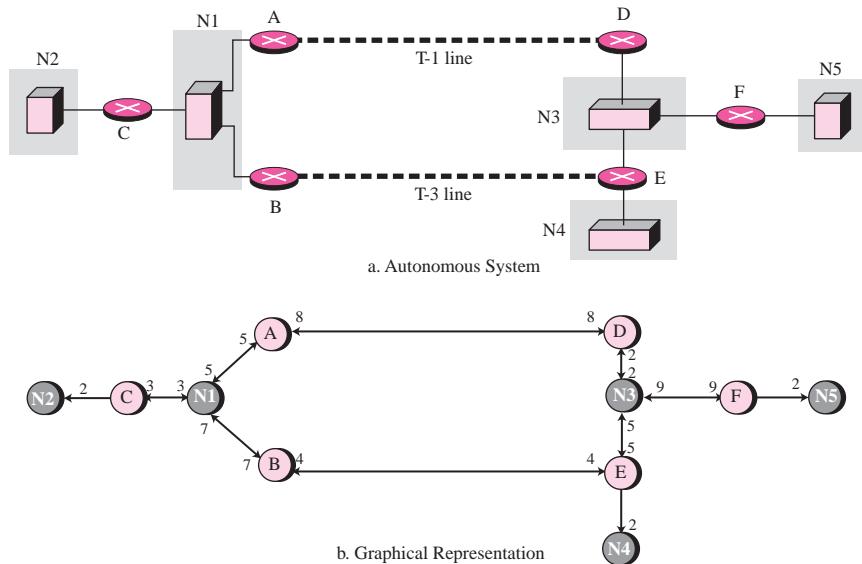
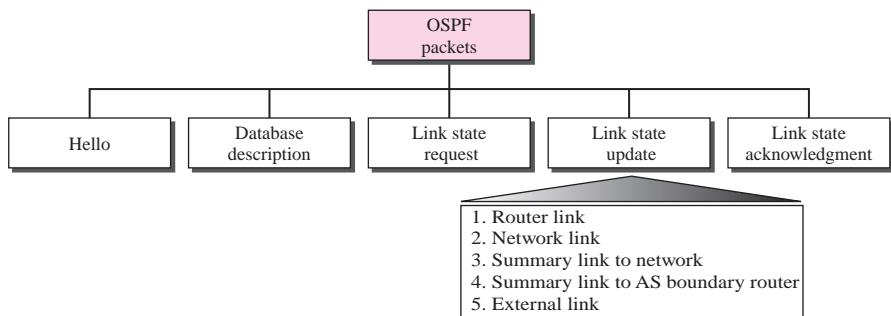
Graphical Representation

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

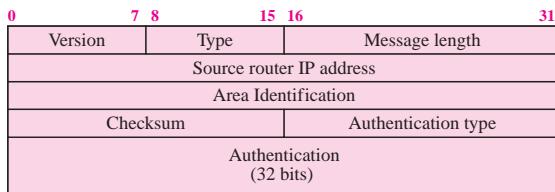
OSPF uses five different types of packets: *hello*, *database description*, *link state request*, *link state update*, and *link state acknowledgment* (see Figure 11.27). The most important one is the link state update that itself has five different kinds.

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.

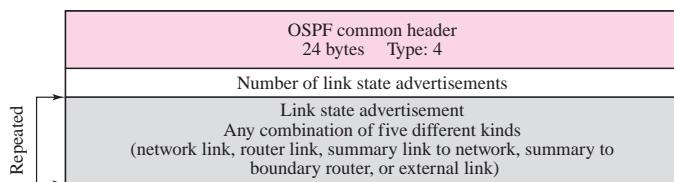
- ❑ **Version.** This 8-bit field defines the version of the OSPF protocol. It is currently version 2.
- ❑ **Type.** This 8-bit field defines the type of the packet. As we said before, we have five types, with values 1 to 5 defining the types.
- ❑ **Message length.** This 16-bit field defines the length 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

Link state age	Reserved	E	T	Link state type
Link state ID				
Advertising router				
Link state sequence number				
Link state checksum	Length			

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.

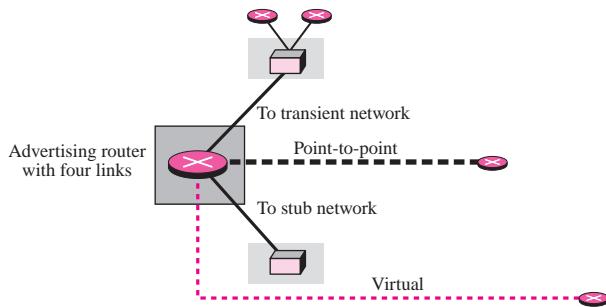
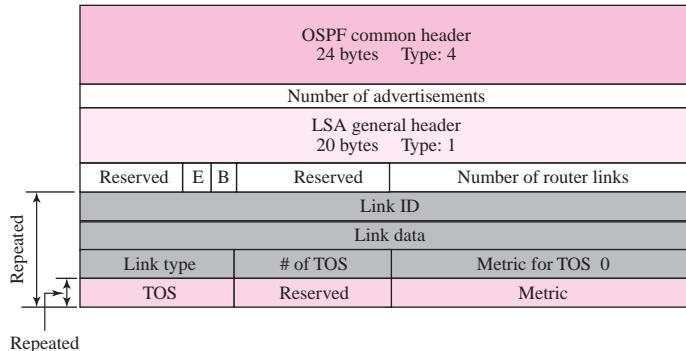
Router Link LSA

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).

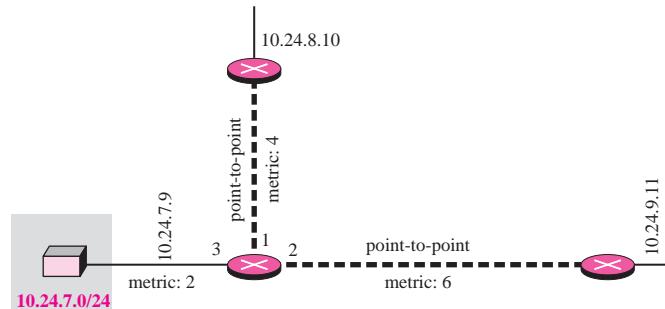
Figure 11.31 Router link**Figure 11.32** Router link LSA**Table 11.5** Link Types, Link Identification, and Link Data

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

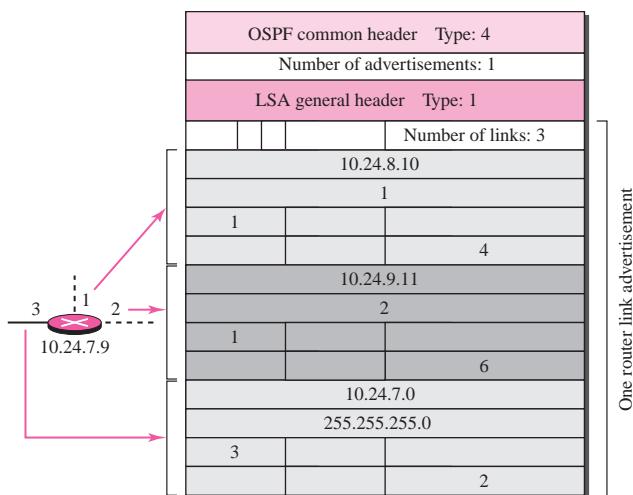
- ❑ **Link type.** Four different types of links are defined based on the type of network to which the router is connected (see Table 11.5).
- ❑ **Number of types of service (TOS).** This field defines the number of types of services announced for each link.
- ❑ **Metric for TOS 0.** This field defines 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

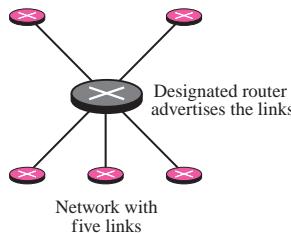
Give the router link LSA sent by router 10.24.7.9 in Figure 11.33.

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.

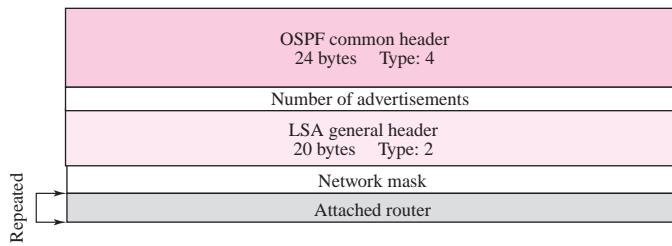
Figure 11.34 Solution to Example 11.8**Network Link LSA**

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

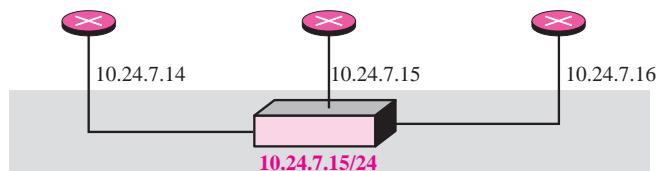
network link advertisement is shown in Figure 11.36. The fields of the network link LSA are as follows:

- ❑ **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

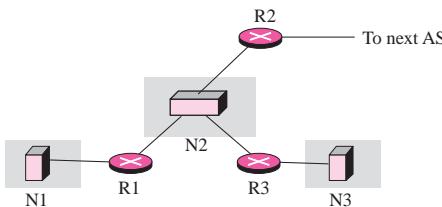
The network for which the network link advertises has three routers attached. The LSA shows the mask and the router addresses. Figure 11.38 shows the network link LSA.

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.

- R1 has two links, N1 and N2.
- R2 has one link, N1.
- 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:

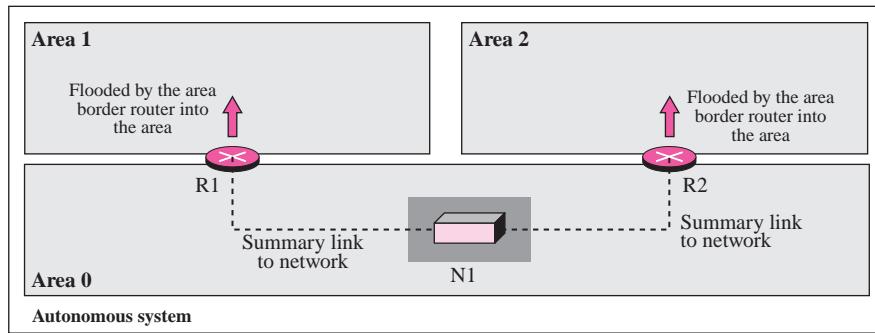
- Advertisement for N1 is done by R1 because it is the only attached router and therefore the designated router.
- Advertisement for N2 can be done by either R1, R2, or R3, depending on which one is chosen as the designated router.
- Advertisement for N3 is done by R3 because it is the only attached 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

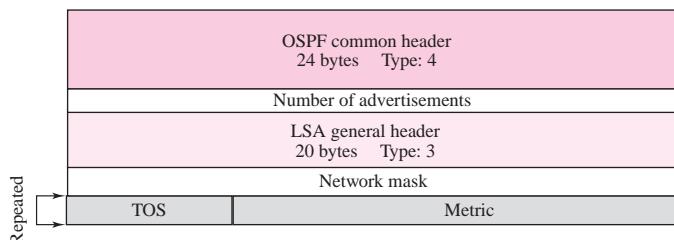
advertisements, and, as we will see, creates a routing table for each area. For example, in Figure 11.40, router R1 is an area border router. It has two routing tables, one for area 1 and one for area 0. R1 floods area 1 with information about how to reach a network located in area 0. In the same way, 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. The 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

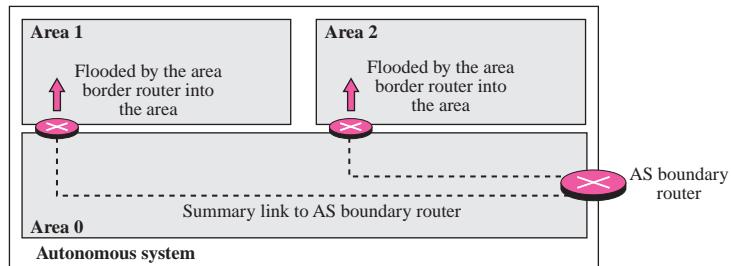
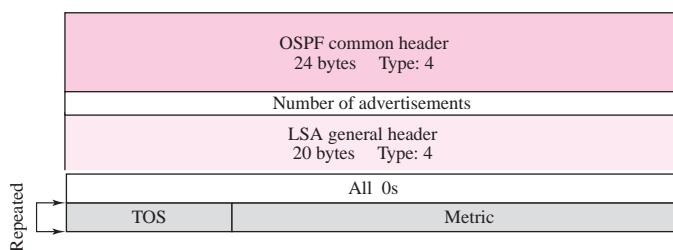


Figure 11.43 Summary link to AS boundary router LSA



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

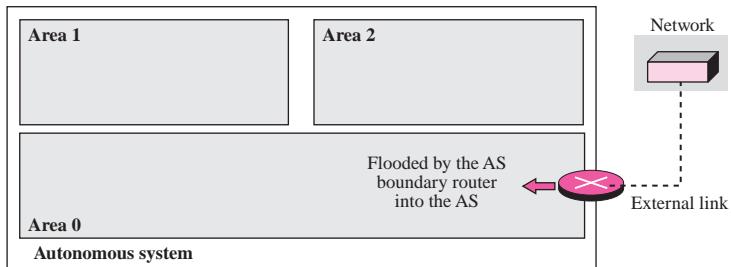
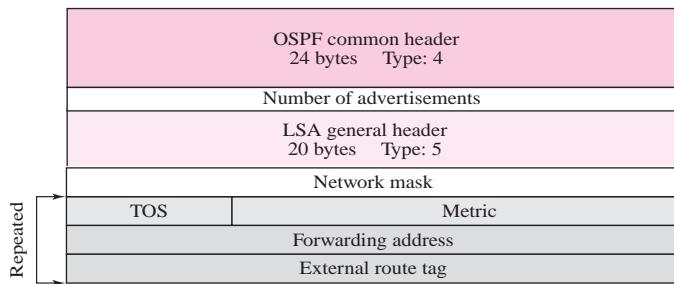


Figure 11.45 External link LSA

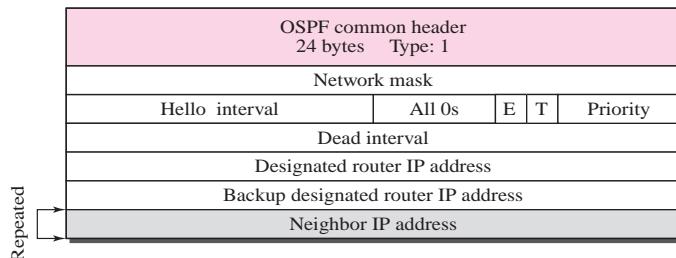


Other Packets

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

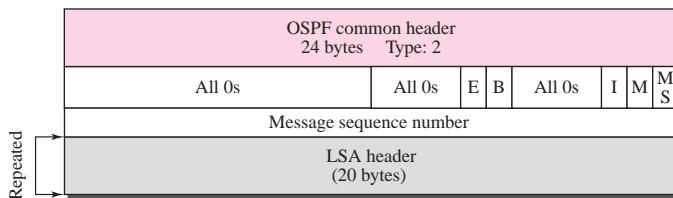
- ❑ **Network mask.** This 32-bit field defines the network mask of the network over which the hello message is sent.
- ❑ **Hello interval.** This 16-bit field defines the number of seconds between hello messages.
- ❑ **E flag.** This is a 1-bit flag. When it is set, it means that the area is a stub 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:

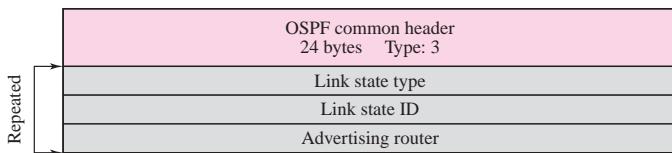
Figure 11.47 Database description packet



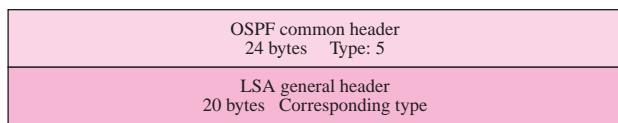
- ❑ **E flag.** This 1-bit flag is set to 1 if the advertising router is an autonomous boundary router (*E* stands for external).
- ❑ **B flag.** This 1-bit flag is set to 1 if the advertising router is an area border router.
- ❑ **I flag.** This 1-bit field, the *initialization* flag, is set to 1 if 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.

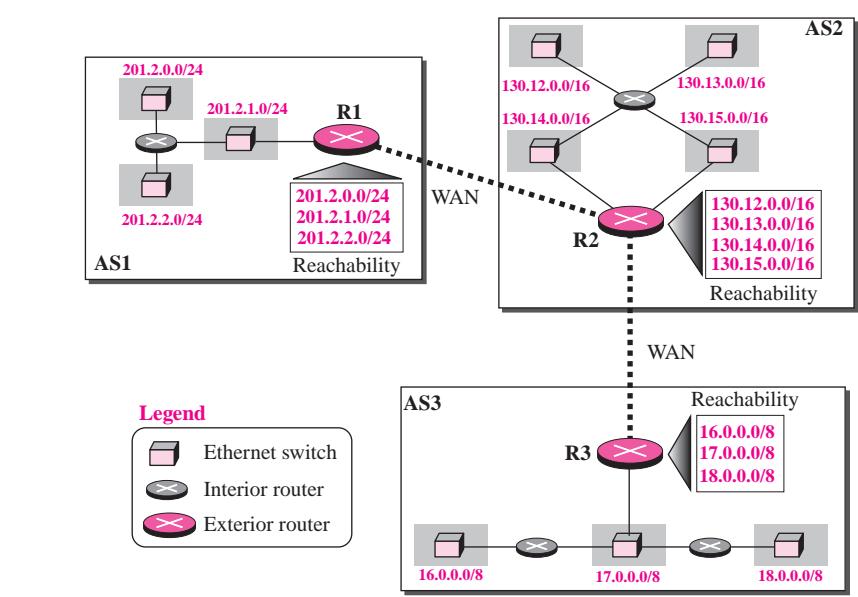
Example 11.11

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
R2
R3

Path-Vector Routing Table
Path-Vector Routing Table
Path-Vector Routing Table

Network
Path
Network
Path
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

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

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)

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

R1		R2		R3	
Network	Path	Network	Path	Network	Path
201.2.0.0/22	AS1 (This AS)	201.2.0.0/22	AS2, AS1	201.2.0.0/22	AS3, AS2, AS1
130.12.0.0/18	AS1, AS2	130.12.0.0/18	AS2 (This AS)	130.12.0.0/18	AS3, AS2
16.0.0.0/6	AS1, AS2, AS3	16.0.0.0/6	AS2, AS3	16.0.0.0/6	AS3 (This AS)
Path-Vector Routing Table		Path-Vector Routing Table		Path-Vector Routing Table	

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

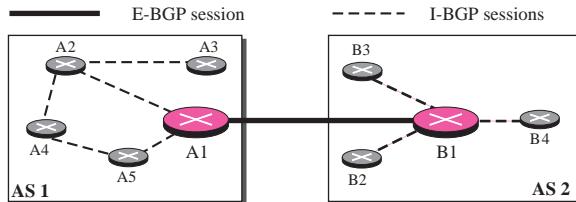
The exchange of routing information between two routers using BGP takes place in a session. A session is a connection that is established between two BGP routers only for the sake of exchanging routing information. To create a reliable environment, BGP uses the services of TCP. In other words, a session 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

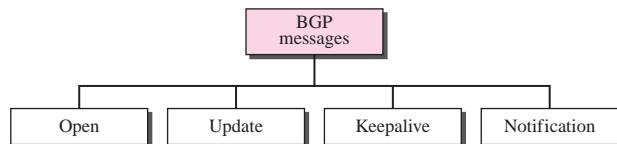


The session established between AS1 and AS2 is an E-BGP session. The two speaker routers exchange information they know about networks in the Internet. However, these two routers need to collect information from other routers in the autonomous systems. This is done using I-BGP sessions.

Types of Packets

BGP uses four different types of messages: **open**, **update**, **keepalive**, and **notification** (see Figure 11.54).

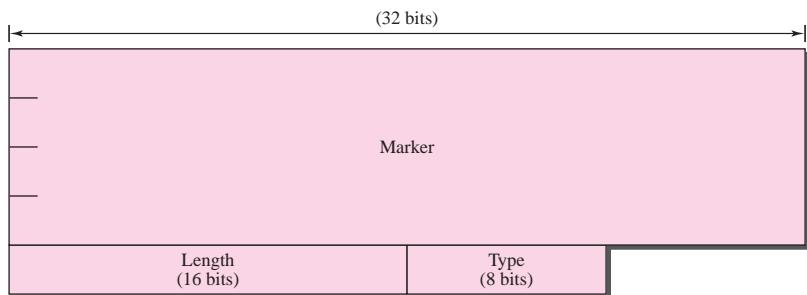
Figure 11.54 Types of BGP messages



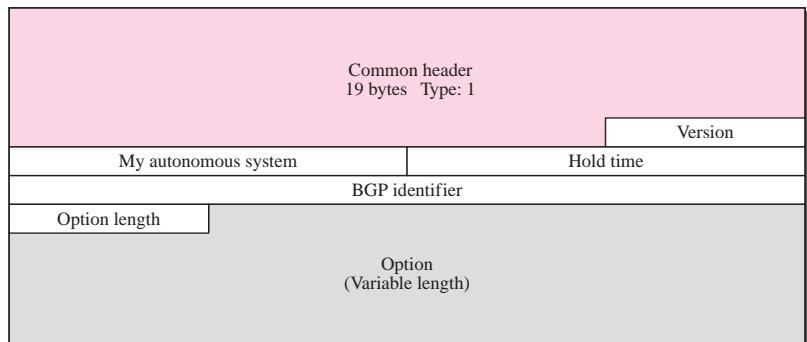
Packet Format

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:

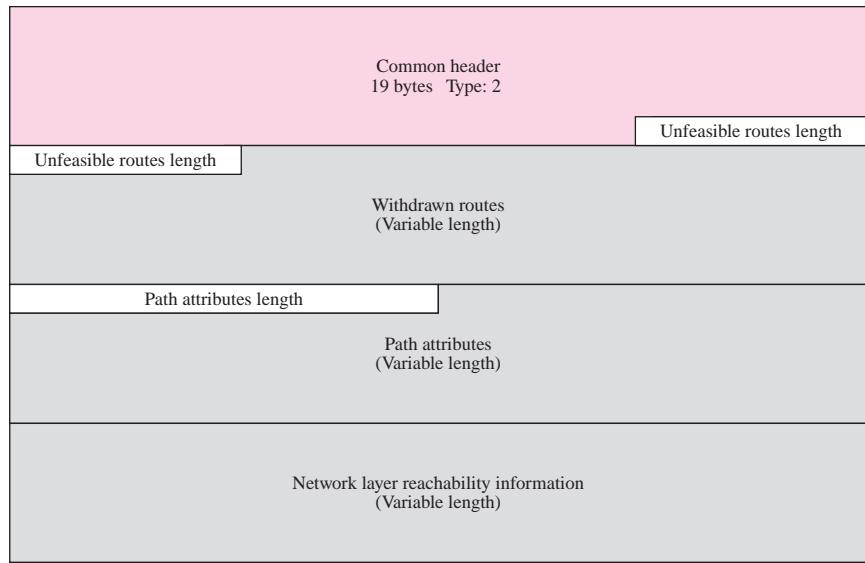
- ❑ **Version.** This 1-byte field defines the version of BGP. The current version is 4.
- ❑ **My autonomous system.** This 2-byte field defines the autonomous system number.
- ❑ **Hold time.** This 2-byte field defines the maximum number of seconds that can elapse until one of the parties receives a 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.

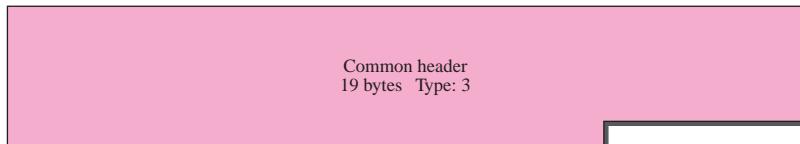
- ❑ **Path attributes length.** This 2-byte field defines the length of the next field.
- ❑ **Path attributes.** This field defines the attributes of the path (route) to the network whose reachability is being announced in this message.
- ❑ **Network layer reachability information (NLRI).** This field 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.

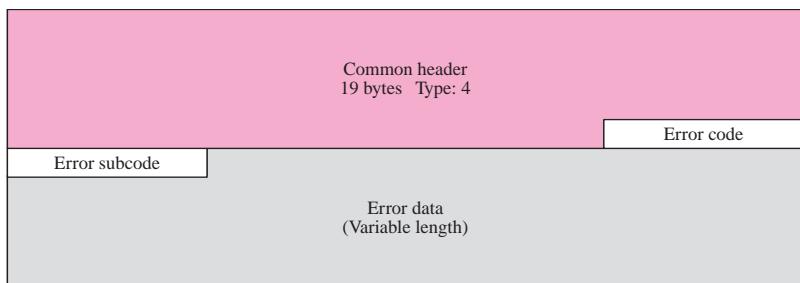
Figure 11.58 *Keepalive message*



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	link state request packet
area border router	link state routing
autonomous system (AS)	link state update packet
autonomous system boundary router	metric
backbone router	notification message
Bellman-Ford algorithm	open message
Border Gateway Protocol (BGP)	Open Shortest Path First (OSPF)
cost	optional attribute
count to infinity	path vector routing
Dijkstra algorithm	periodic timer
distance vector routing	point-to-point link
expiration timer	poison reverse
flooding	Routing Information Protocol (RIP)
garbage collection timer	split horizon
hello interval	stub link
hello message	transient link
hop count	update message
inter-domain routing	virtual link
intra-domain routing	well-known attribute
keepalive message	

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.

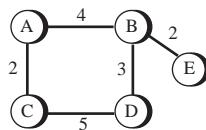
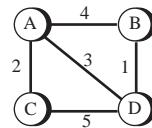
8. The router in Exercise 7 receives a RIP message with four records from router C as shown below:

(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:
 - Show the link state update/router link advertisement for router A.
 - Show the link state update/router link advertisement for router D.
 - Show the link state update/router link advertisement for router E.
 - Show the link state update/network link advertisement for network N2.
 - Show the link state update/network link advertisement for network N4.
 - 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.

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

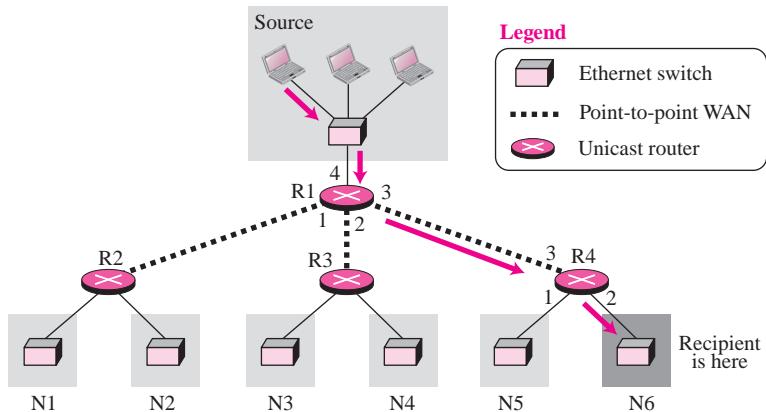
From the previous chapters in this part of the book, we have learned that forwarding a datagram by a router is normally based on the prefix of the destination address in the datagram, which defines the network to which the destination host is connected. Of course, address aggregation mechanism may combine 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*



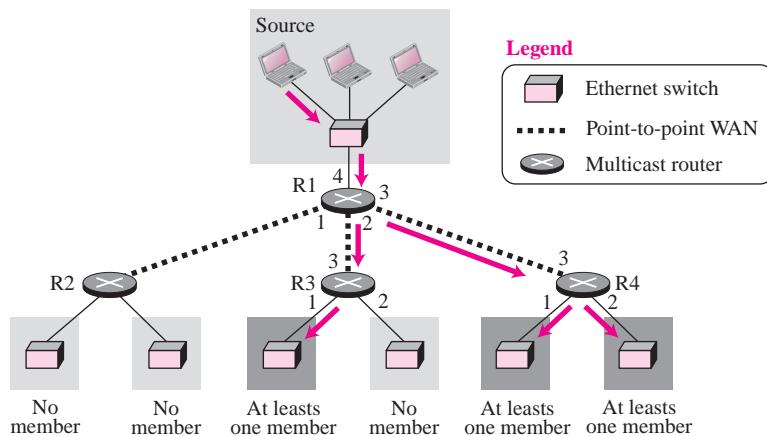
In unicasting, the routing table that defines the only output port for each datagram, is based on the optimum path as we saw in Chapter 11.

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 multi-cast 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



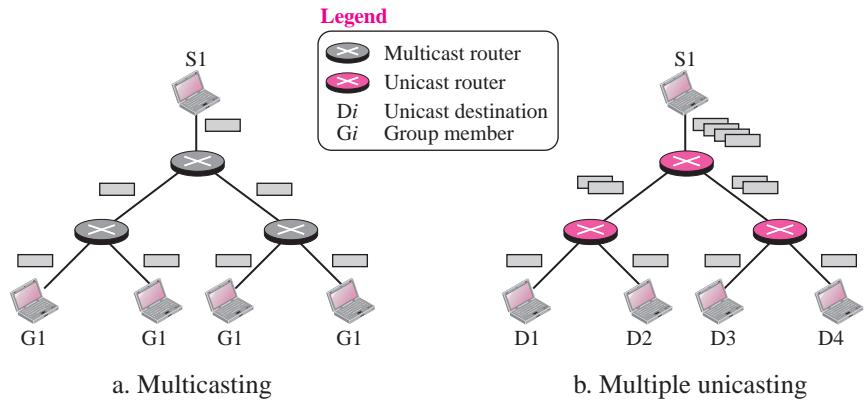
In multicasting, a multicast router may have to send out copies of the same datagram through more than one interface. In Figure 12.2, router R1 needs to send out the datagram through interfaces 2 and 3. Similarly, router R4 needs to send out the datagram through both its interfaces. Router R3, however, 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



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	Internet 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.

Internet Control Block

The block 224.0.1/24 is called the Internet 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 used 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	10
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.

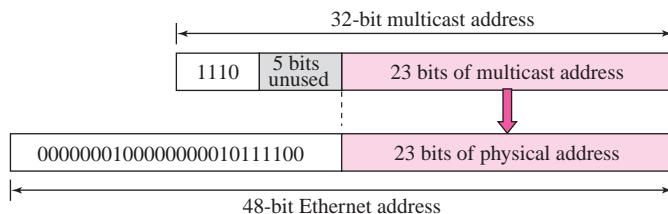
Delivery of Multicast Packets at Data Link Layer

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 IPv4 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

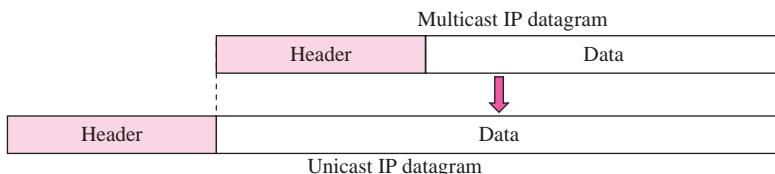
- a. The rightmost 3 bytes in hexadecimal are D4:18:09. We need to subtract 8 from the leftmost digit, resulting in 54:18:09.
- b. We add the result of part a to the Ethernet multicast starting address. The result is

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.

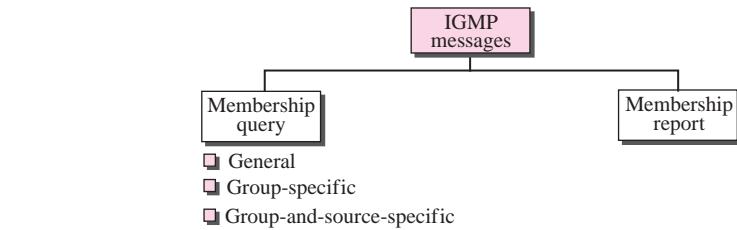
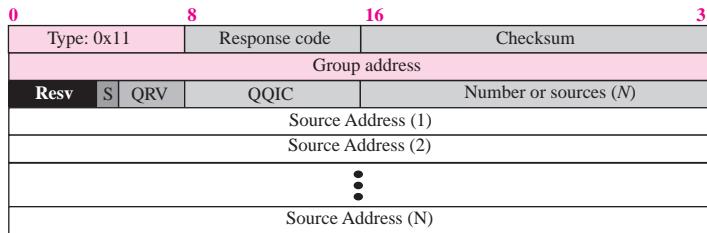
IGMP has gone through three versions. Versions 1 and 2 provide what is called any-source multicast (ASM), which means that the group members receive a multicast message no matter where it comes from. The IGMP version 3 provides what is called source-specific multicast (SSM), which means that the recipient 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

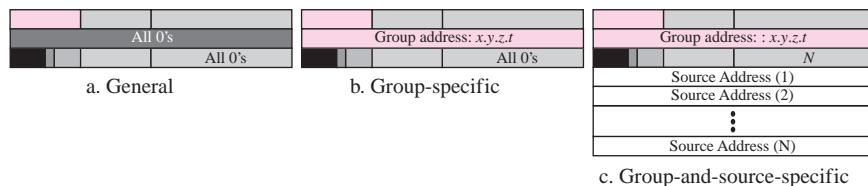
A brief description of each field follows:

- ❑ **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 *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

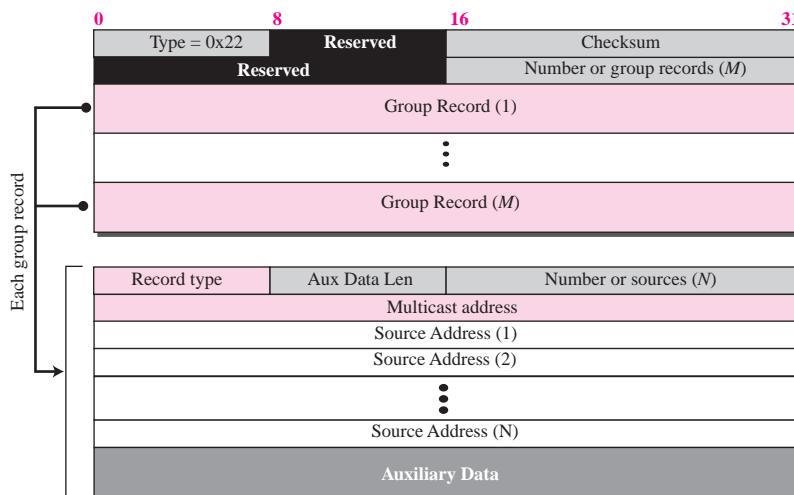


- a. In a *general query message*, the querier router probes each neighbor to report the whole list of its group membership (interest in any multicast group).
- b. In a *group-specific query message*, the querier router probes each neighbor to report if it is still interested in a specific multicast 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

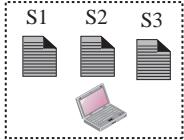
The management of groups starts with the processes (running application programs) on a host connected to an interface. Each process, which is associated with a *socket* as we will see in Chapter 17, has a record for each multicast group from which the socket wishes to receive a multicast message. The 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 change 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.

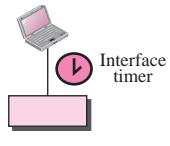
$$\text{exclude source list} = \{a, b, c\} \cdot \{b, c, g\} - \{a, b, d, e\} = \{c\}$$

- 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

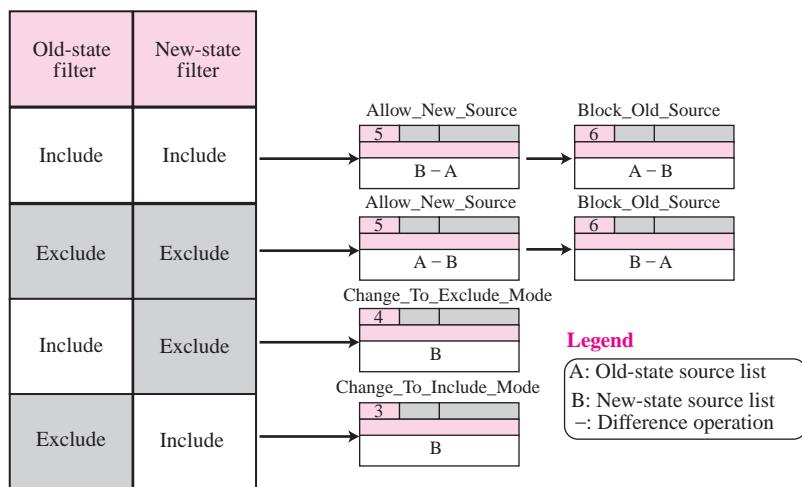
Sending Change-State Reports

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 $(A \cdot B)$ 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.

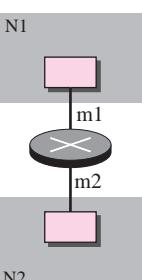
IGMP Protocol Applied to Router

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

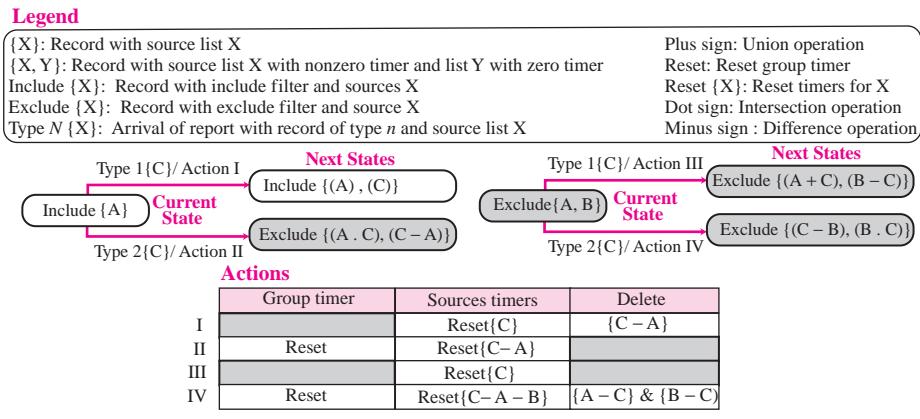
Multicast group	Timer	Filter	Source addresses
226.10.11.8	⌚	Exclude	(b, ⌚)
227.21.25.41	⌚	Include	(a, ⌚) (b, ⌚) (c, ⌚)
228.32.12.40	⌚	Include	(d, ⌚) (e, ⌚) (f, ⌚)

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



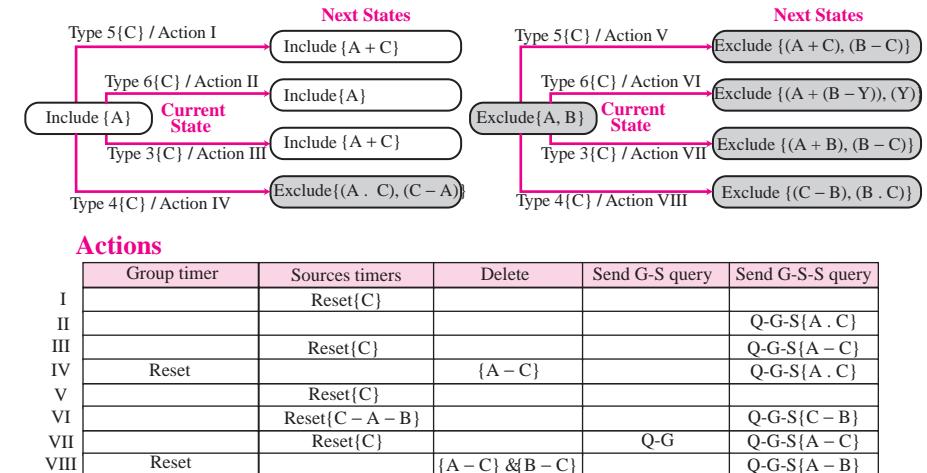
Reception of Reports in Response to Other Queries When a router sends a group-specific or group-and-source-specific query, it expects to receive a report or reports. This type of report normally contains Filter_Mode_Change_Record (types 3 and 4) or Source_List_Change Record (types 5 and 6). When such 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**Legend**

$\{X, Y\}$: Record with sources X with nonzero timer and sources Y with zero timer	Plus (+): Union operation
Include $\{X\}$: Record with include filter and sources X	$\{X\}$: Record with sources X
Exclude $\{X\}$: Record with exclude filter and source X	Q-G: Send group-specific query
Type N $\{X\}$: Arrival of report with record of type N and sources X	Reset: Reset group timer
Q-G-S $\{X\}$: Send group-and-source-specific query with sources X	Reset $\{X\}$: Reset timers for source X
Dot (.) : Intersection operation	Minus (-): Difference operation



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.

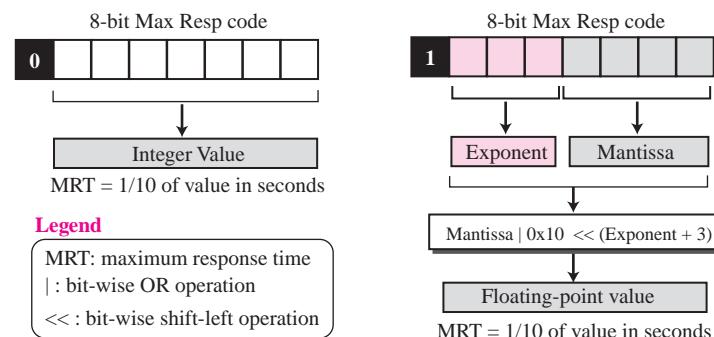
Variables and Timers

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*

<i>Message Type</i>	<i>IP Address</i>
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*

<i>Version</i>	<i>Type Value</i>	<i>Message Type</i>
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.

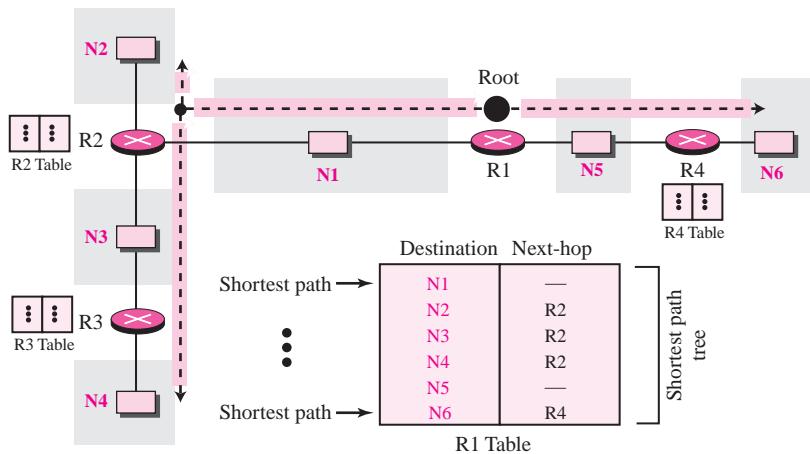
Unicast Routing

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.

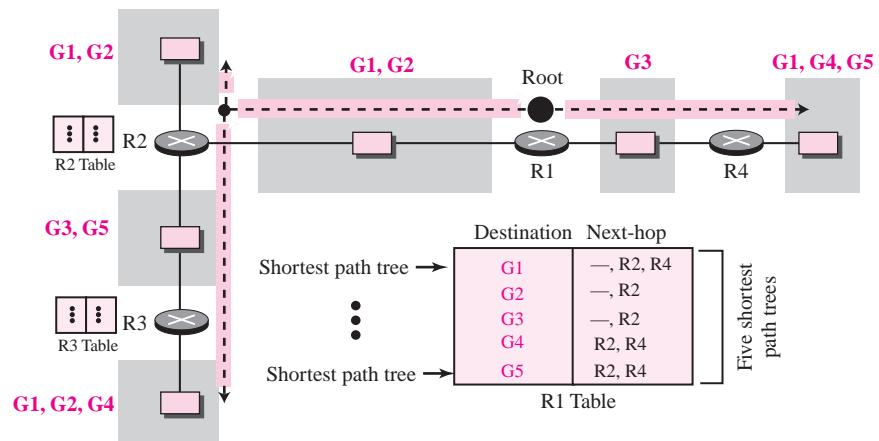
In multicast routing, each involved router needs to construct a shortest path tree for each group.

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

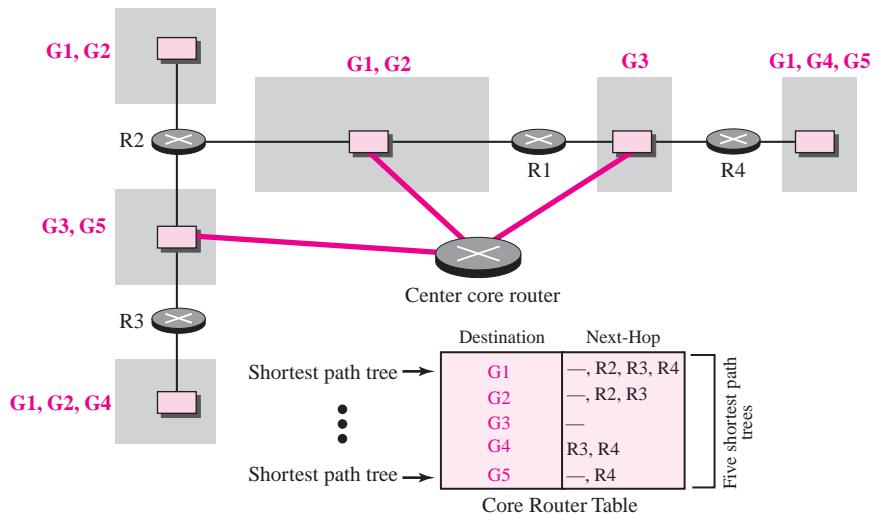


Group-Shared Tree In the **group-shared tree** approach, instead of each router having m shortest path trees, only one designated router, called the center core, or rendezvous router, takes the responsibility of distributing multicast traffic. The core has m shortest path trees in its routing table. 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

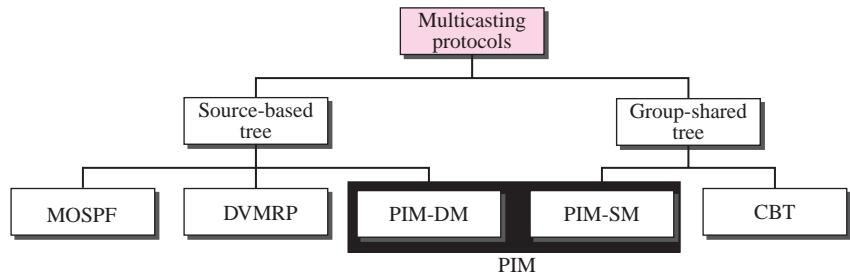
During the last few decades, several multicast routing protocols have emerged. Some of these protocols are extensions of unicast routing protocols; some are totally new. We discuss these protocols in the remainder of this chapter. Figure 12.21 shows the taxonomy of these protocols.

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.

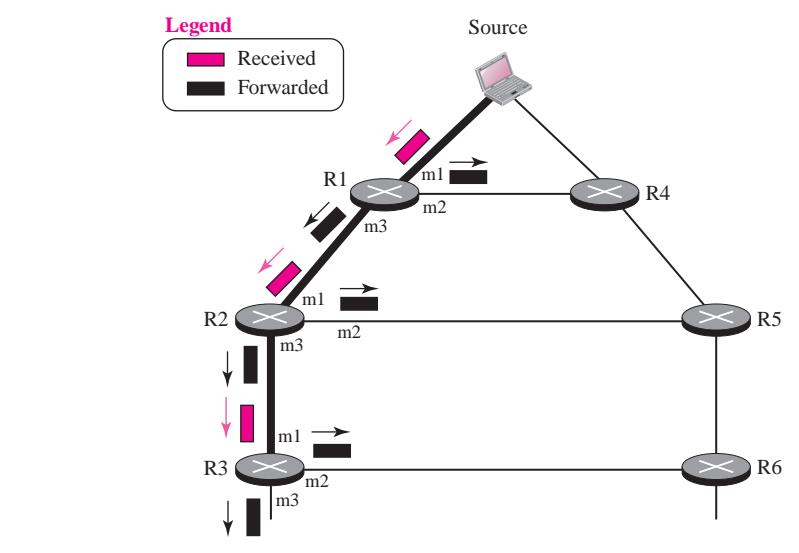
To accomplish this, the multicast distance vector algorithm uses a process based on four decision-making strategies. Each strategy is built on its predecessor. We explain them one by one and see how each strategy can improve the shortcomings of the previous one.

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.

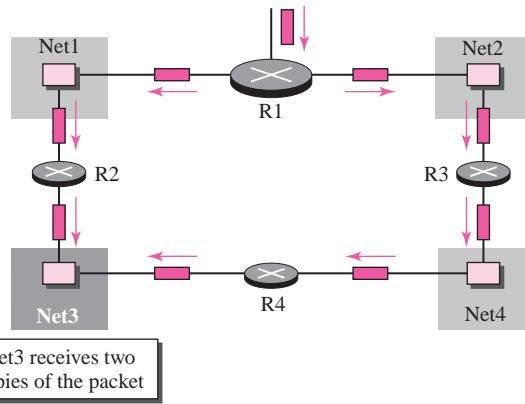
Figure 12.22 RPF

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)

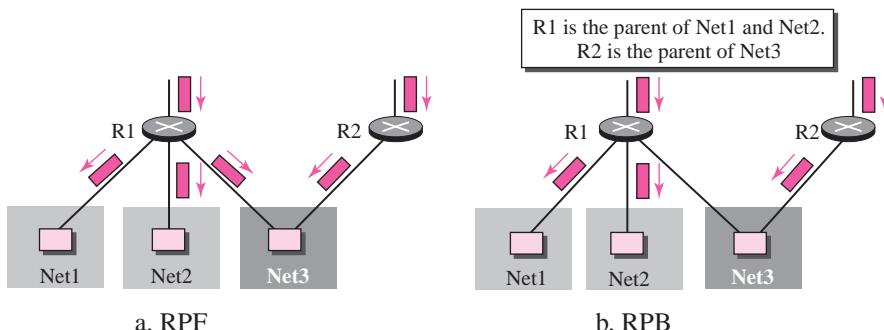
RPF guarantees that each network receives a copy of the multicast packet without formation of loops. However, RPF does not guarantee that each network receives only one copy; a network may receive two or more copies. The reason is that RPF is not based on the destination address (a group address); forwarding 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

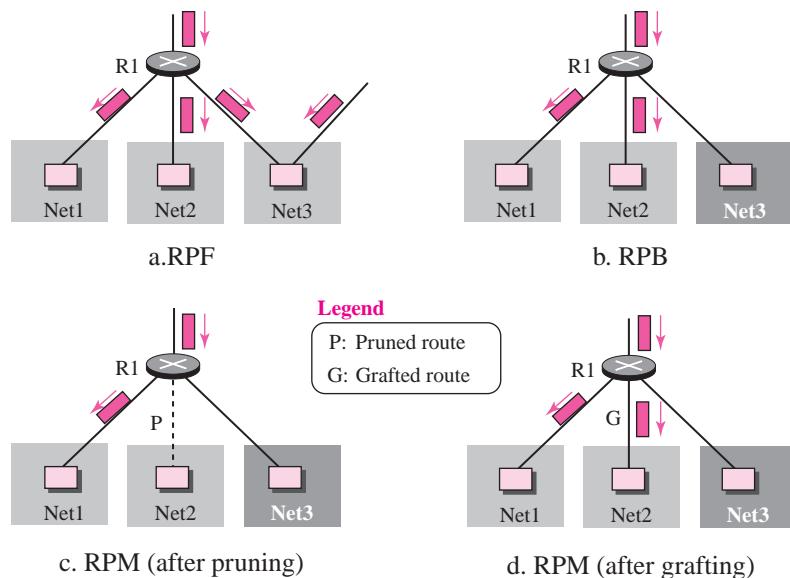
neighborhood has the shortest path to the source (when interpreting the source as the destination). If more than one router qualifies, the router with the smallest IP address is selected.

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

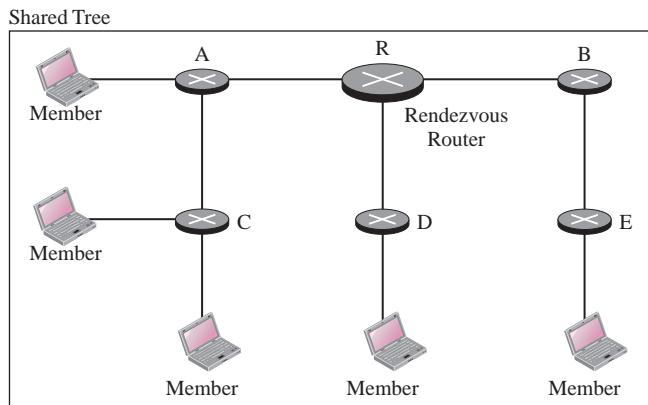
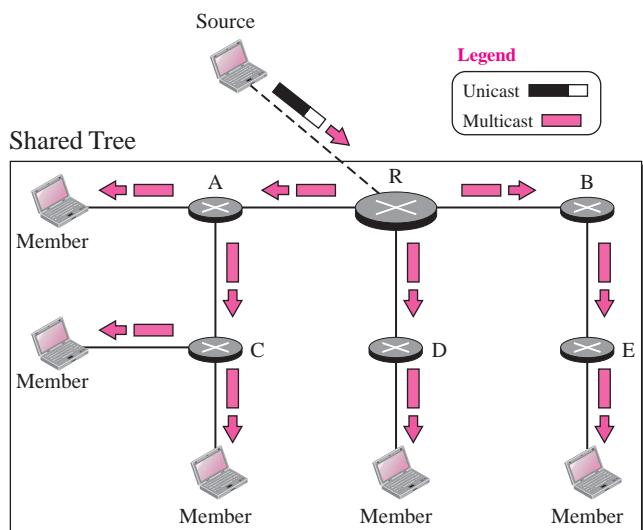
After the rendezvous point is selected, every router is informed of the unicast address of the selected router. Each router with an intercreated group then sends a unicast join message (similar to a grafting message) to show that it wants to join the group. This message passes through all routers that 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/grafiting 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/grafiting 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.

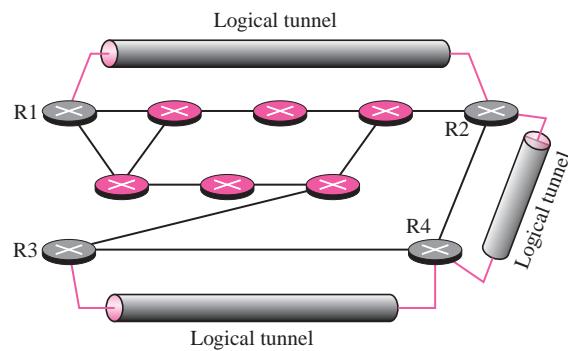
One of the characteristics of PIM-SM is that it can switch from a group-shared tree strategy to a source-based tree strategy when necessary. This can happen if there is a dense area of activity far from the RP. That area can be more efficiently handled with a source-based tree strategy instead of a group-shared tree strategy.

PIM-SM is similar to CBT but uses a simpler procedure.

12.6 MBONE

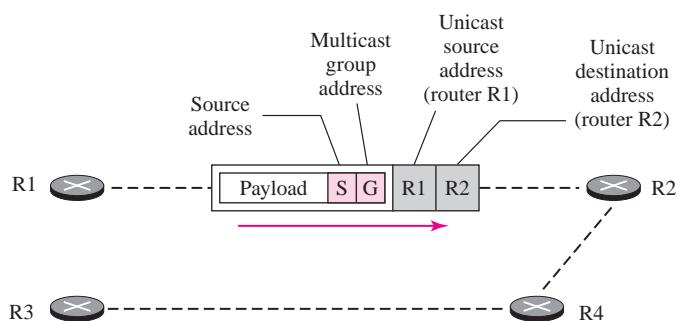
Multimedia and real-time communication have increased the need for multicasting in the Internet. However, only a small fraction of Internet routers are multicast routers. In other words, a multicast router may not find another multicast router in the neighborhood to forward the multicast packet. Although 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

flooding

data-driven

graft message

Distance Vector Multicast Routing Protocol
(DVMRP)

group-shared tree

Internet Group Management Protocol (IGMP)

multicast backbone (MBONE)	prune message
Multicast Open Shortest Path First (MOSPF)	reverse path broadcasting (RPB)
Protocol Independent Multicast (PIM)	reverse path forwarding (RPF)
Protocol Independent Multicast, Dense Mode (PIM-DM)	reverse path multicasting (RPM)
Protocol Independent Multicast, Sparse Mode (PIM-SM)	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.

P A R T

3

Transport Layer

Chapter 13 Introduction to the Transport Layer 374

Chapter 14 User Datagram Protocol (UDP) 414

Chapter 15 Transmission Control Protocol (TCP) 432

Chapter 16 Stream Control Transmission Protocol (SCTP) 502

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

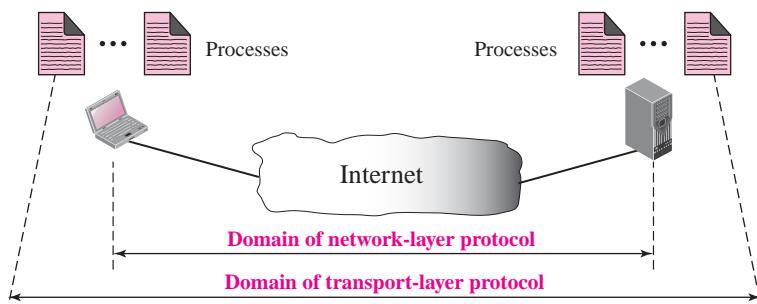
As we discussed in Chapter 2, the transport layer is located between the network layer and the application layer. The transport layer is responsible for providing services to the application layer; it receives services from the network layer. In this section, we discuss the services that can be provided 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

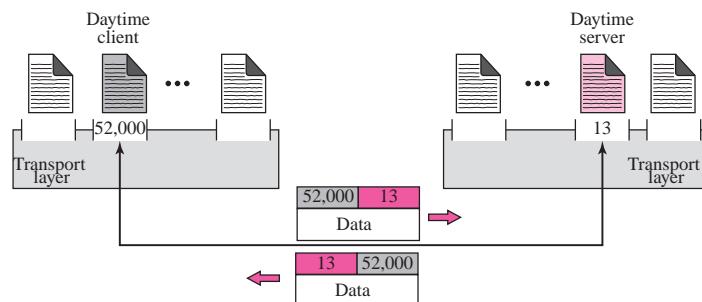
- Local host
- Local process
- Remote host
- Remote process

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.

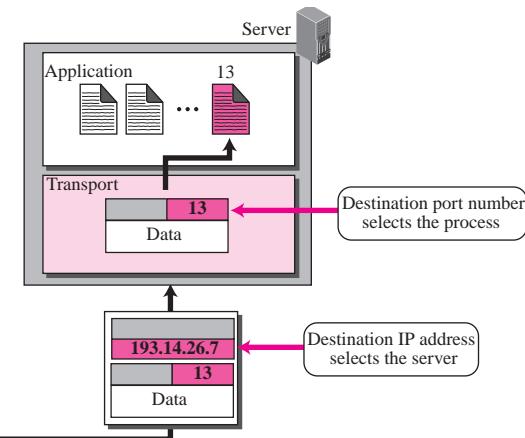
The server process must also define itself with a port number. This port number, however, cannot be chosen randomly. If the computer at the server site runs a server process and assigns a random number as the port number, the process at the client site that wants to access that server and use its services 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).

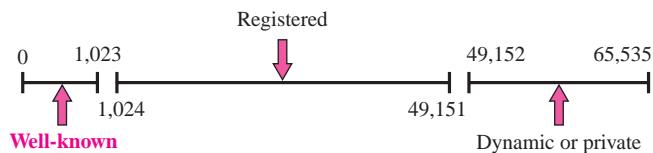
Figure 13.3 IP addresses versus port numbers



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.

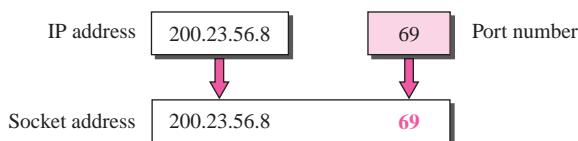
```
$grep tftp /etc/services
tftp      69/tcp
tftp      69/udp
```

```
$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*

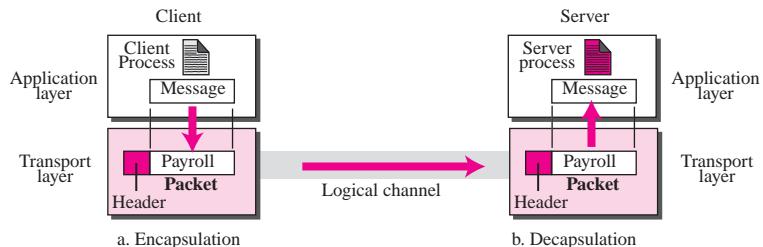


To use the services of transport layer in the Internet, we need a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the network-layer packet header and the transport-layer packet header. The first header contains the IP addresses; 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.

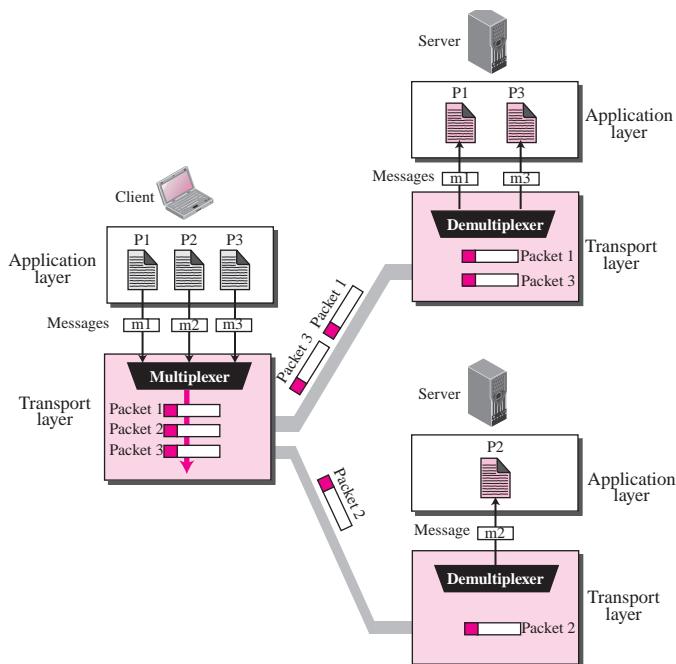
Multiplexing and Demultiplexing

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.

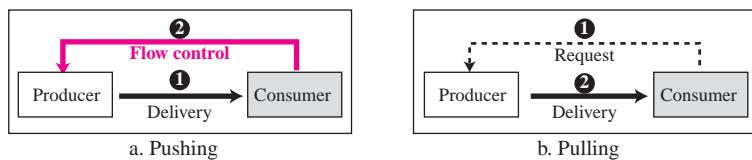
Flow Control

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

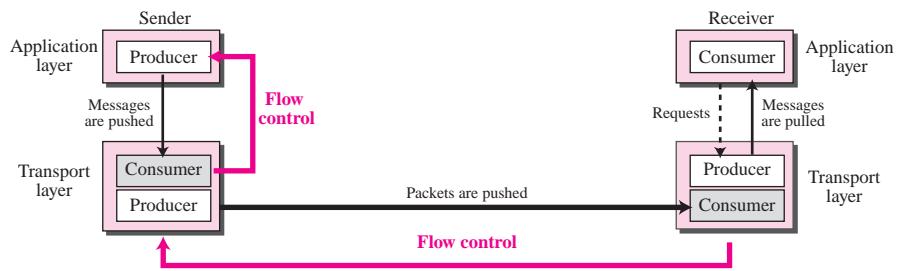
When the producer *pushes* the items, the consumer may be overwhelmed and there is a need for flow control, in the opposite direction, to prevent the discarding of the items. In other words, the consumer needs to warn the producer to stop the delivery and to inform it when it is ready again to receive 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.

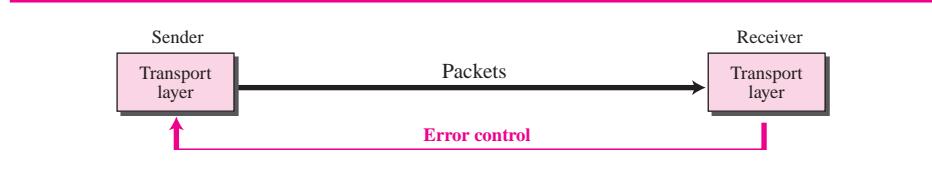
Error Control

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.

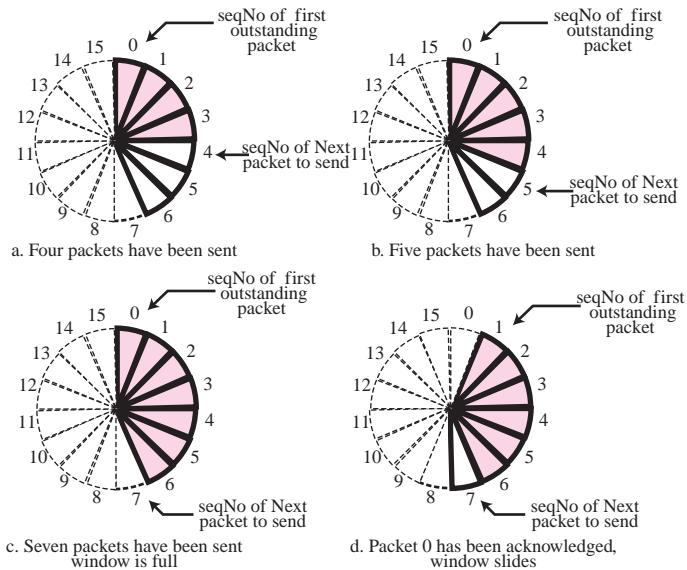
At the sender, when a packet is prepared to be sent, we use the number of the next free location, x , in the buffer as the sequence number of the packet. When the packet is sent, a copy is stored at memory location x , awaiting the acknowledgment from the other end. When an acknowledgment related to 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

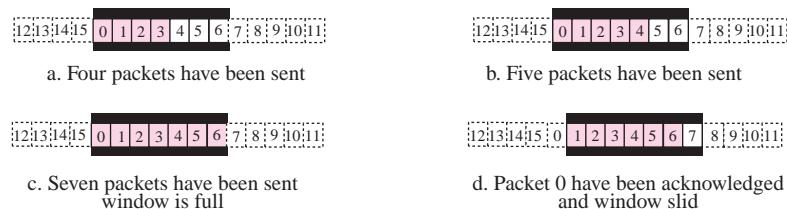
Since the sequence numbers used modulo 2^m , a circle can represent the sequence number from 0 to $2^m - 1$ (Figure 13.11).

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.

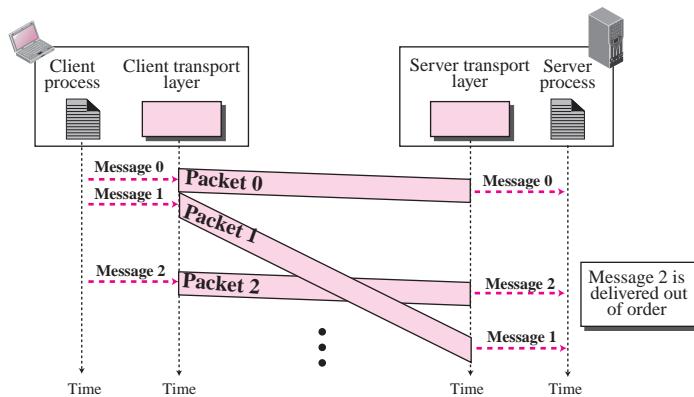
Connectionless and Connection-Oriented Services

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.

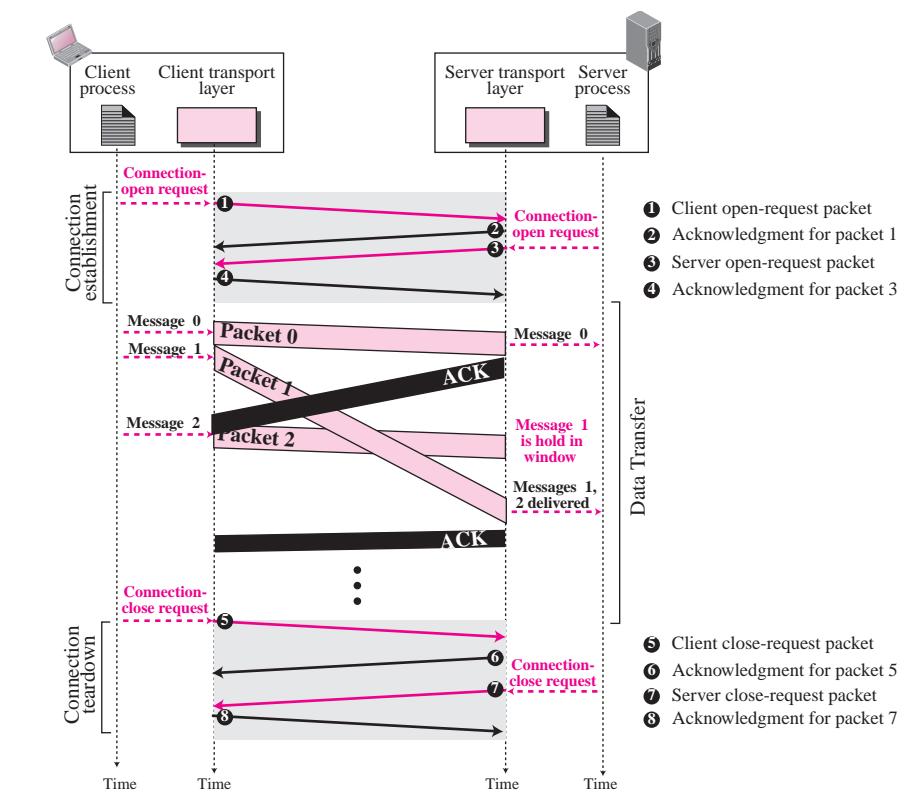
The above two problems arise from the fact that the two transport layers do not coordinate with each other. The receiving transport layer does not know when the first packet will come nor when all of the packets have arrived.

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 torn 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



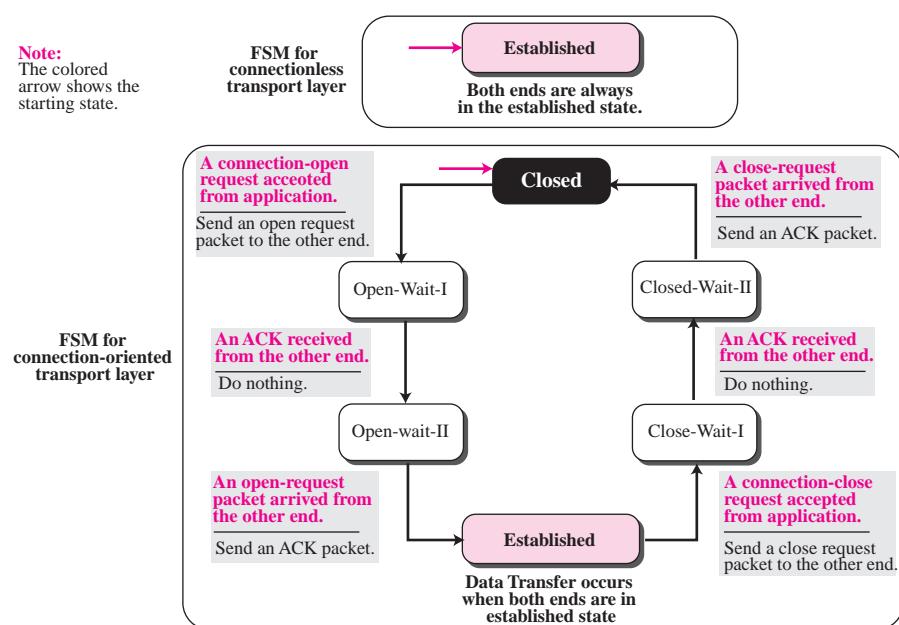
service means a coordination between the two end hosts and all the routers in between. At the transport layer, connection-oriented service involves only the two hosts; the service is end to end. This means that we should be able to make a connection-oriented protocol over either a connectionless or connection-oriented protocol. Figure 13.14 shows the connection establishment, data transfer, and tear-down 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.

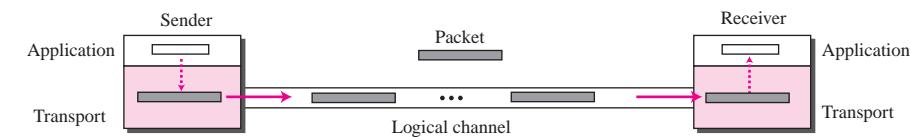
13.2 TRANSPORT-LAYER PROTOCOLS

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

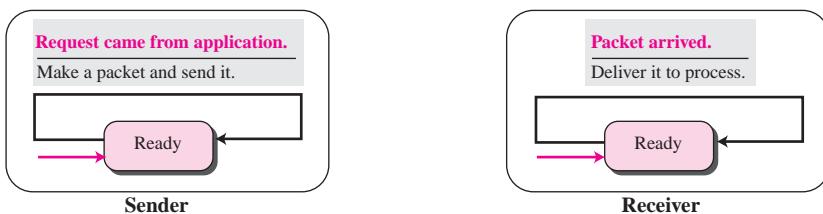


The transport layer at the sender gets a message from its application layer, makes a packet out of it, and sends the packet. The transport layer at the receiver receives a packet from its network layer, extracts the message from the packet, and delivers the message to its application layer. The transport 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

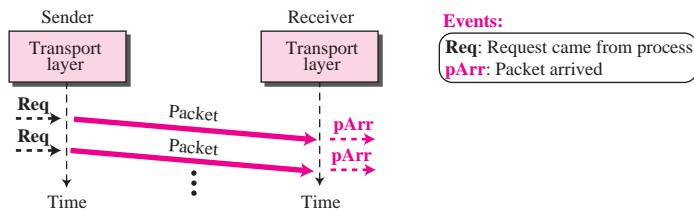


The simple protocol is a connectionless protocol that provides neither flow nor error control.

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.

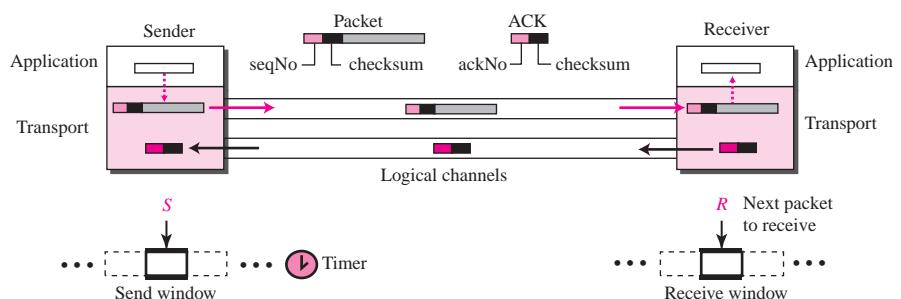
Figure 13.18 Flow diagram for Example 13.3



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.

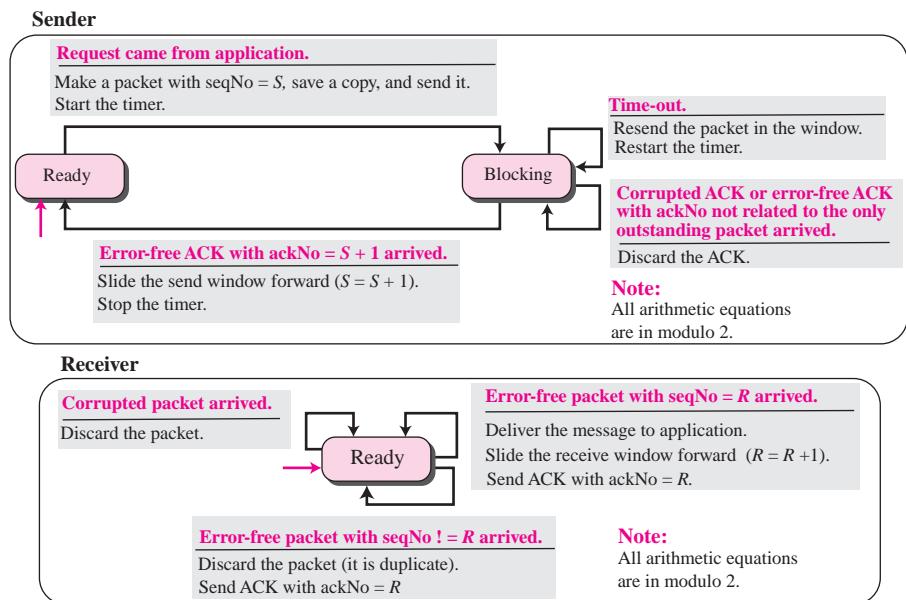
The sender has a control variable, which we call S (sender), that points to the only slot in the send window. The receiver has a control variable, which we call R (receiver), that points to the only slot in the receive window.

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 $ackNo \neq (S + 1)$ modulo 2 arrives, the ACK is discarded.
3. If a time-out occurs, the sender resends the only outstanding packet and restarts the timer.

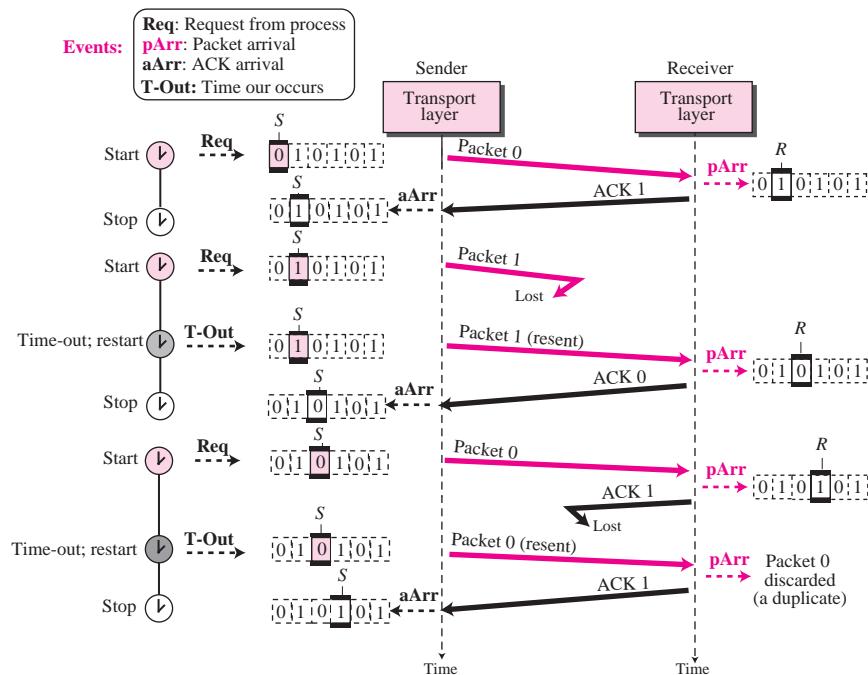
Receiver The receiver is always in the *ready* state. The variable R is initialized to 0. Three events may occur:

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 $seqNo \neq R$ 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

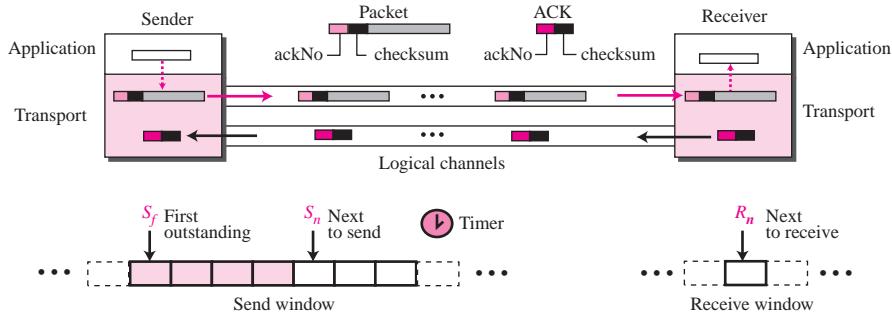
In networking and in other areas, a task is often begun before the previous task has ended. This is known as **pipelining**. There is no pipelining in the Stop-and-Wait protocol because a sender must wait for a packet to reach the destination and be acknowledged before the next packet can be sent. However, 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.

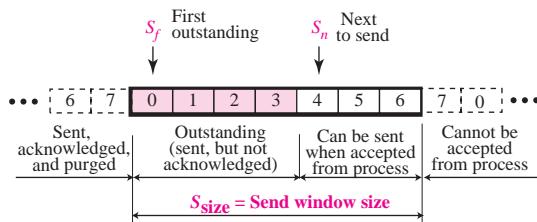
Acknowledgment Number

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.

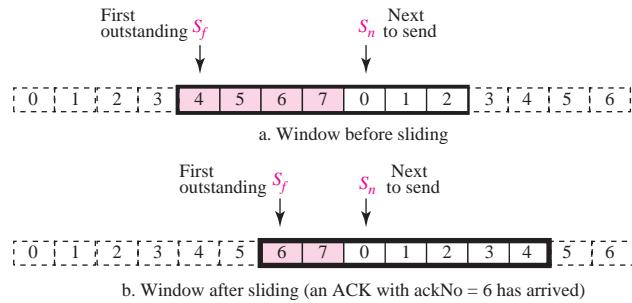
The send window is an abstract concept defining an imaginary box of maximum size = $2^m - 1$ with three variables: S_f , S_n , and S_{size} .

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.

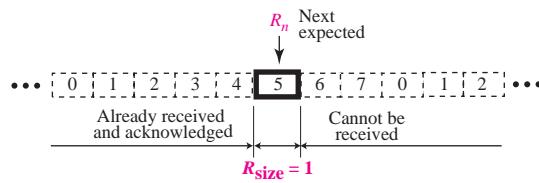
The send window can slide one or more slots when an error-free ACK with $ackNo$ between S_f and S_n (in modular arithmetic) arrives.

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, $R_n = (R_n + 1) \bmod 2^m$.

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

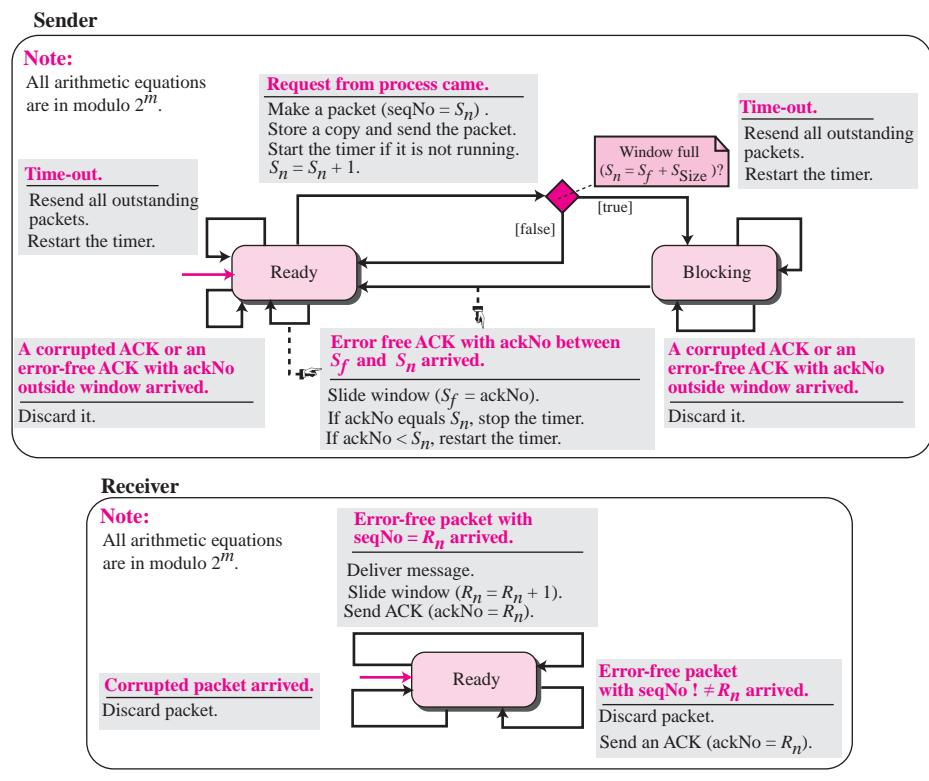
When the timer expires, the sender resends all outstanding packets. For example, suppose the sender has already sent packet 6 ($S_n = 7$), but the only timer expires. If $S_f = 3$,

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, $S_n = (S_f + S_{\text{size}})$ 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 $S_f = \text{ackNo}$) and if all outstanding packets are acknowledged ($\text{ackNo} = S_n$), 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 $S_f = \text{ackNo}$) and if all outstanding packets are acknowledged ($\text{ackNo} = S_n$), 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 $\text{seqNo} = R_n$ arrives, the message in the packet is delivered to the application layer. The window then slides, $R_n = (R_n + 1) \bmod 2^m$. Finally an ACK is sent with $\text{ackNo} = R_n$.
2. If an error-free packet with seqNo outside the window arrives, the packet is discarded, but an ACK with $\text{ackNo} = R_n$ 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.

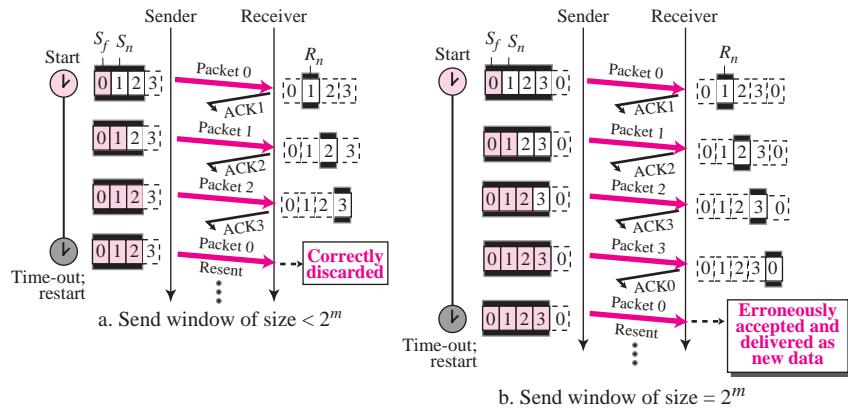
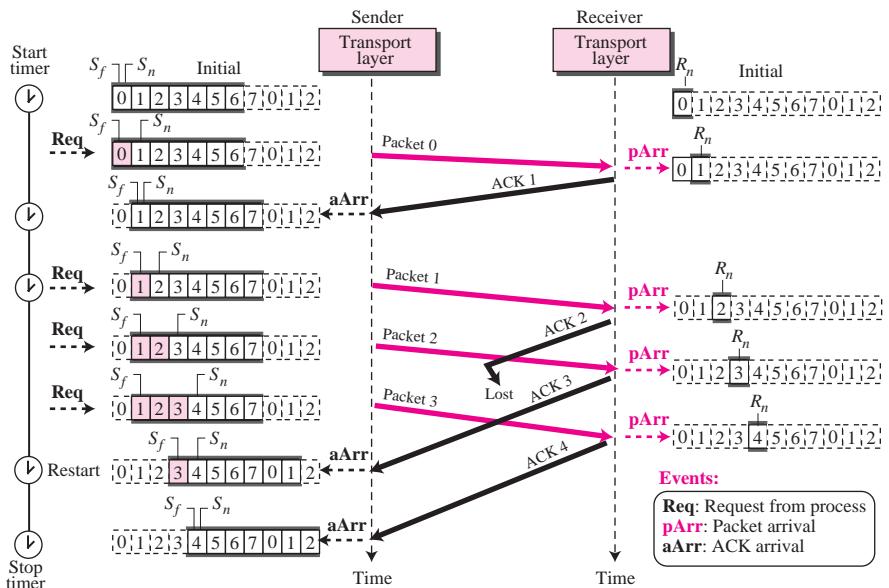
Figure 13.27 Send window size for Go-Back-N**Example 13.7**

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.

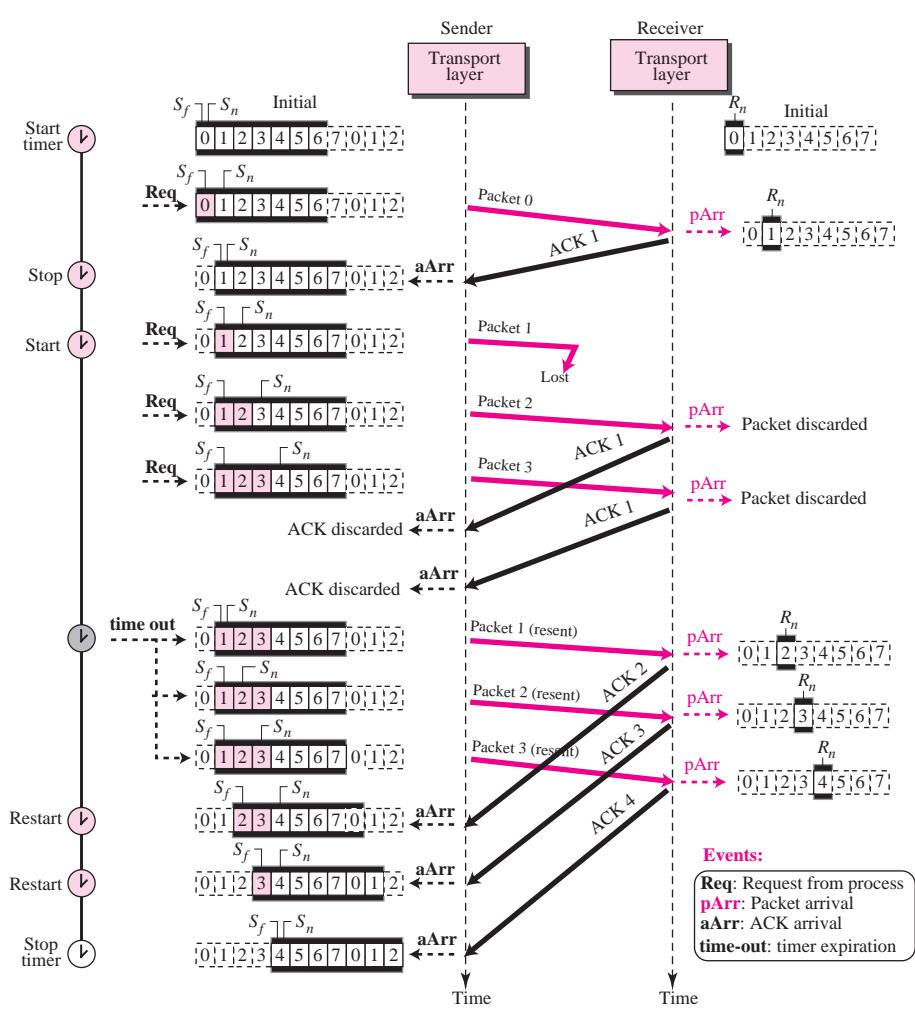
Figure 13.28 Flow diagram for Example 13.7

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 than 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

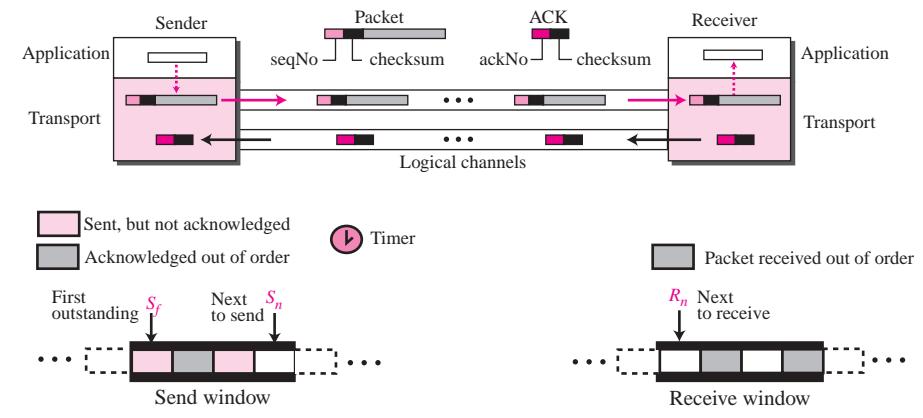
The reader may find that there is a similarity between the Go-Back- N protocol and the Stop-and-Wait protocol. The Stop-and-Wait protocol is actually a Go-Back- N protocol in which there are only two sequence numbers and the send window size is 1. In other words, $m = 1$ and $2^m - 1 = 1$. In Go-Back- 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

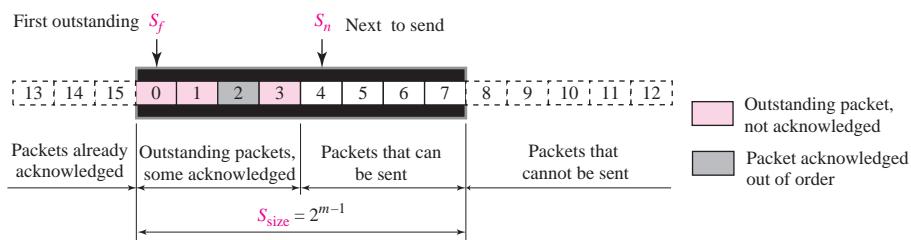


Windows

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.

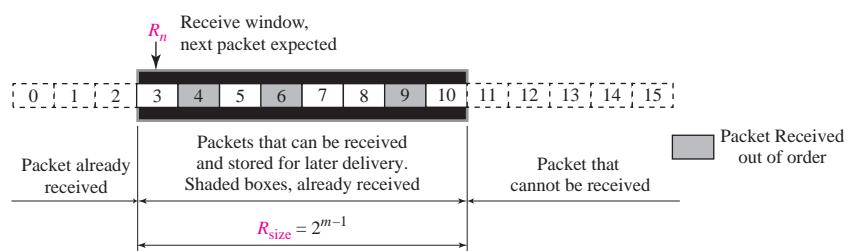
The send window maximum size can be 2^{m-1} . For example, if $m = 4$, the sequence numbers go from 0 to 15, but the maximum size of the window is just 8 (it is 15 in the Go-Back-N Protocol). We show the Selective-Repeat send window in Figure 13.31 to emphasize the size.

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.

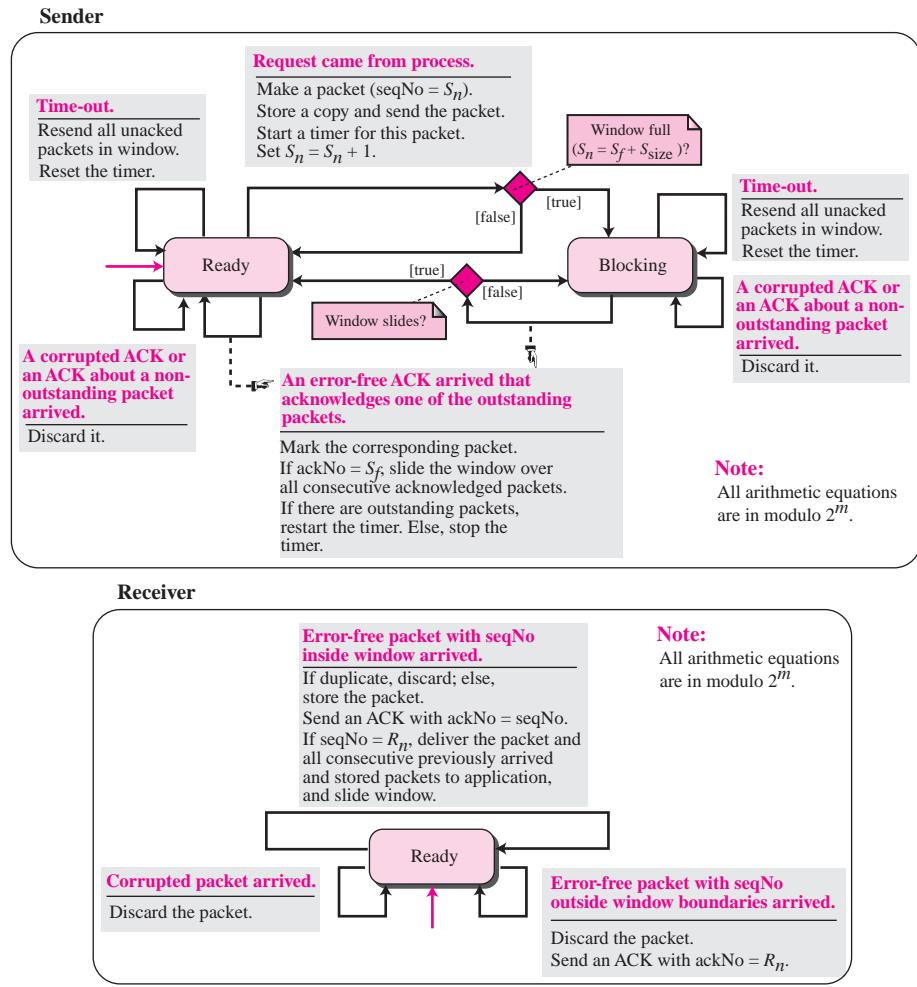
Ready State. Four events may occur in this case:

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, $S_n = (S_n + 1) \bmod 2^m$. If the window is full, $S_n = (S_f + S_{size}) \bmod 2^m$, 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.

- If a corrupted ACK or an error-free ACK with the ackNo not related to outstanding packets arrives, the ACK is discarded.
- 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:

- 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.

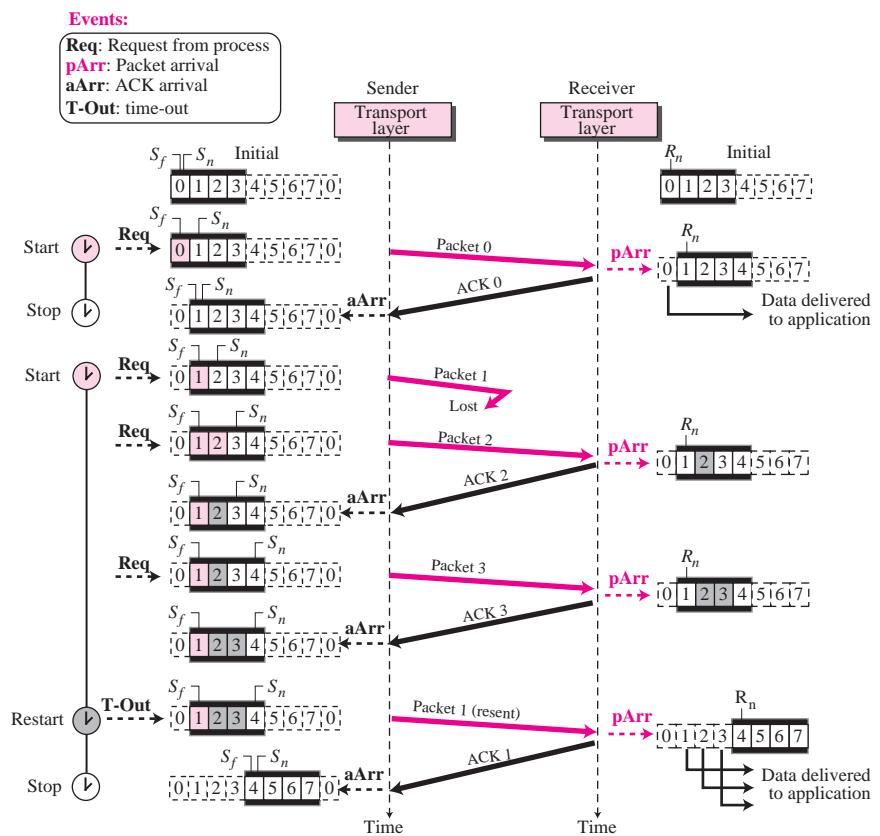
2. If an error-free packet with seqNo outside the window arrives, the packet is discarded, but an ACK with $ackNo = R_n$ is returned to the sender. This is needed to let the sender slide its window if some ACKs related to packets with $seqNo < R_n$ were lost.
3. If a corrupted packet arrives, the packet 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

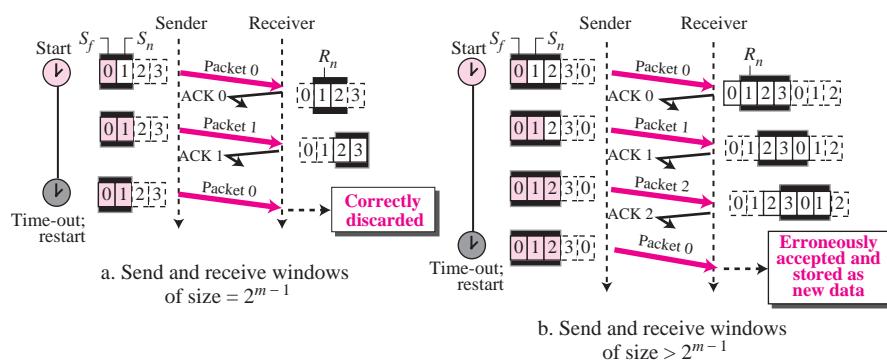


At the receiver site we need to distinguish between the acceptance of a packet and its delivery to the application layer. At the second arrival, packet 2 arrives and is stored and marked (shaded slot), but it cannot be delivered because packet 1 is missing. At the next arrival, packet 3 arrives and is 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 .

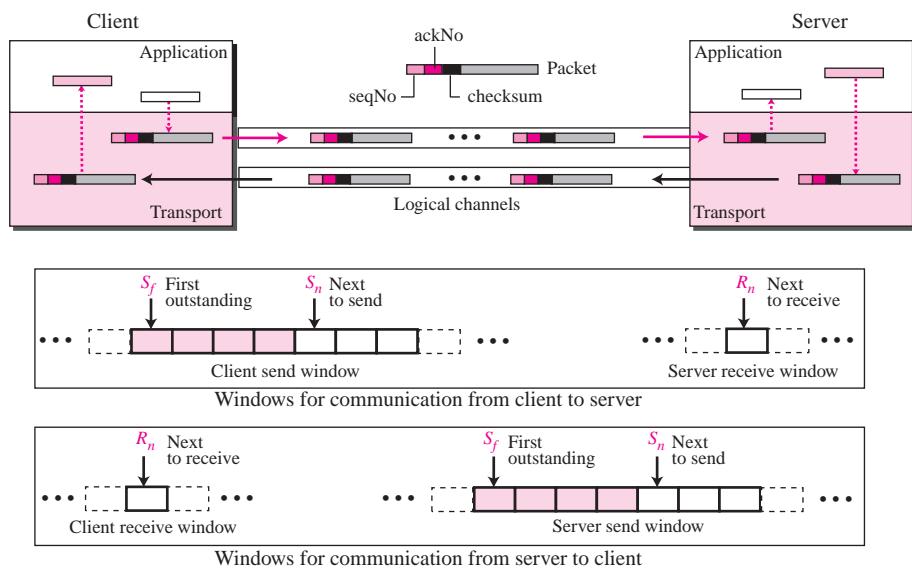
Bidirectional Protocols: Piggybacking

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	process-to-process communication
Go-back-N protocol	Selective-Repeat protocol
load	sequence number
multiplexing	sliding window
open-loop congestion control	socket address
piggybacking	Stop-and-Wait protocol
pipelining	well-known port number
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):
- The sending machine is in the ready state and $S = 0$. What is the sequence number of the next packet to send?
 - 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?
 - 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?
 - 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):
- 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?
 - 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?
 - 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 ?
 - The sending machine is in the blocking state with $S_f = 14$ and $S_n = 21$. What is the size of the window?
 - 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?
 - 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):
- 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?
 - 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?
 - 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?
 - The sending machine is in the blocking state with $S_f = 14$ and $S_n = 21$. What is the size of the window?
 - 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?
 - 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).

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

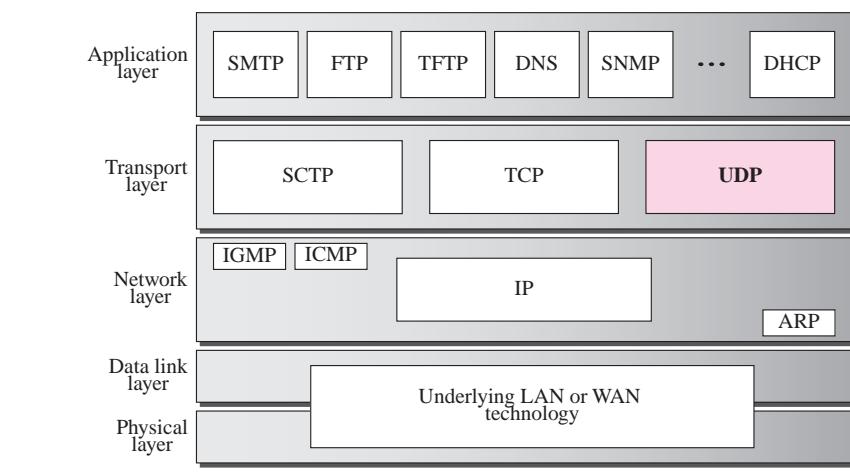
We have several objectives for this chapter:

- ❑ 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.

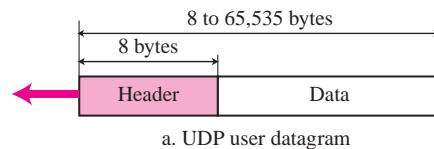
UDP is a **connectionless, unreliable transport protocol**. It does not add anything to the services of IP except for providing process-to-process communication instead of host-to-host communication.

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.

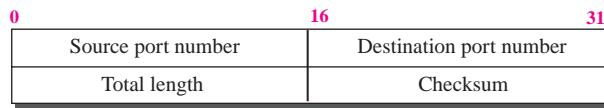
14.2 USER DATAGRAM

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



a. UDP user datagram



b. Header 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.

$$\text{UDP length} = \text{IP length} - \text{IP header's length}$$

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

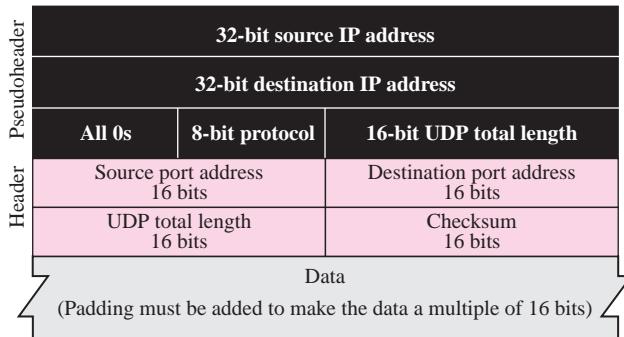
There is **no error control mechanism** in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded. The lack of error control means that the process 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

				10011001 00010010 → 153.18
				00001000 01101001 → 8.105
				10101011 00000010 → 171.2
				00001110 00001010 → 14.10
				00000000 00010001 → 0 and 17
				00000000 00001111 → 15
				00000100 00111111 → 1087
				00000000 00001101 → 13
				00000000 00001111 → 15
				00000000 00000000 → 0 (checksum)
				01010100 01000101 → T and E
				01010011 01010100 → S and T
				01001001 01001110 → I and N
				01000111 00000000 → G and 0 (padding)
				10010110 11101011 → Sum
				01101001 00010100 → Checksum

Example 14.3

What value is sent for the checksum in one of the following hypothetical situations?

- The sender decides not to include the checksum.
- The sender decides to include the checksum, but the value of the sum is all 1s.
- The sender decides to include the checksum, but the value of the sum is all 0s.

Solution

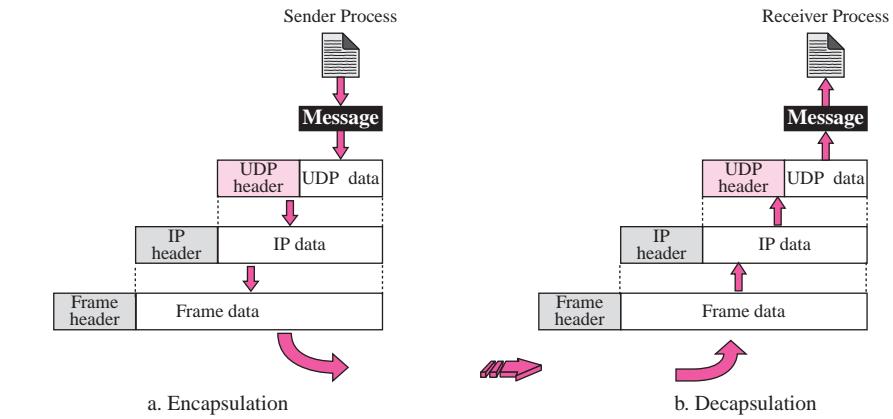
- The value sent for the checksum field is **all 0s** to show that the checksum is **not calculated**.
- When the sender complements the sum, the result is all 0s; the sender complements the result again before sending. The value sent for the checksum is **all 1s**. The second complement operation is needed to avoid confusion with the case in part a.
- 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.

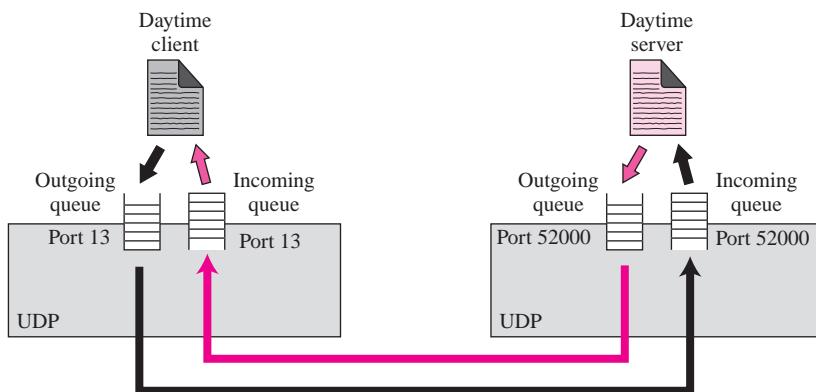
Decapsulation

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.

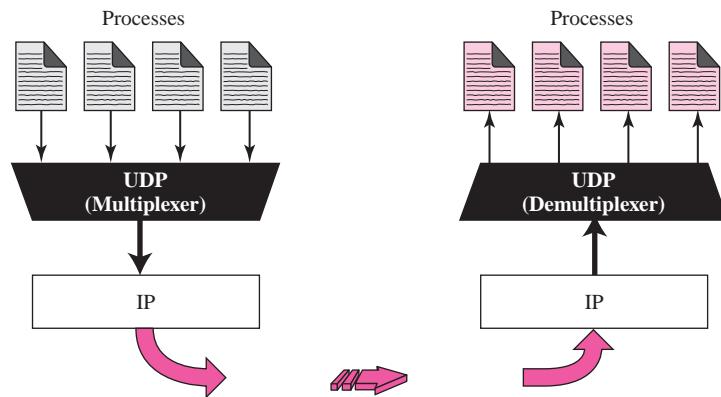
When a message arrives for a server, UDP checks to see if an **incoming queue** has been created for the port number specified in the destination port number field of the user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. If there is no such queue, **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

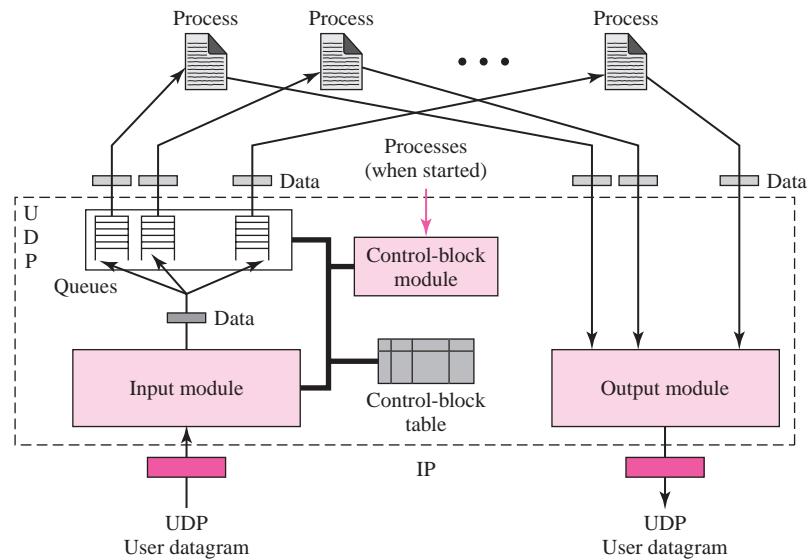
In our package, UDP has a control-block table to keep track of the open ports. Each entry in this table has **a minimum of four fields: the state, which can be FREE or IN-USE, the process ID, the port number, and the corresponding queue number**.

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	user datagram
pseudoheader	User Datagram Protocol (UDP)
queue	

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?

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

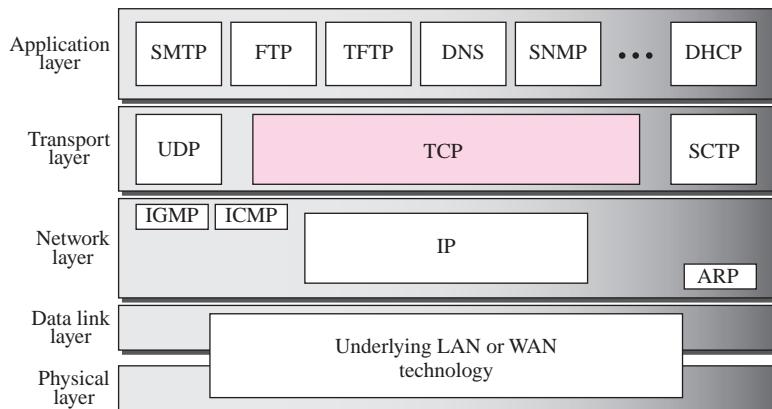
The chapter has several objectives:

- ❑ 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

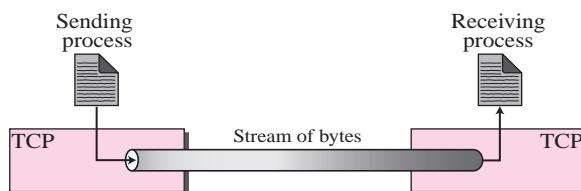
Table 15.1 Well-known Ports used by TCP (continued)

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

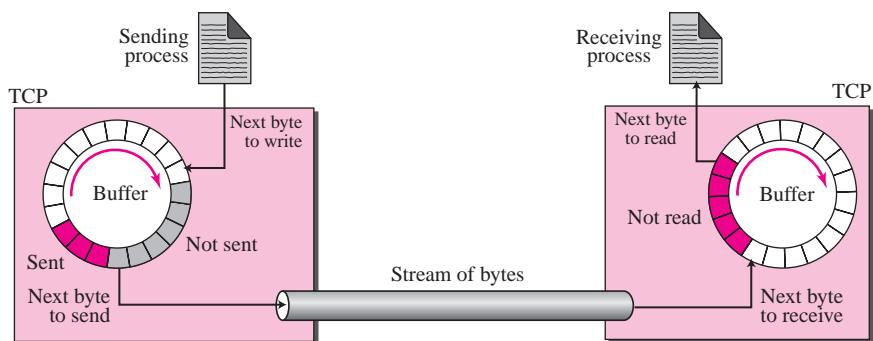
TCP, unlike UDP, is a **stream-oriented protocol**. In UDP, a process sends messages with predefined boundaries to UDP for delivery. UDP adds its own header to each of these messages and delivers it to IP for transmission. Each message from the process is called a **user datagram**, and becomes, eventually, 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

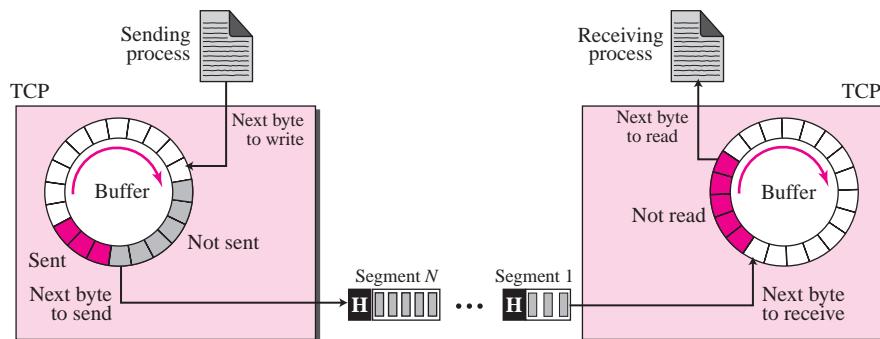
The figure shows the movement of the data in one direction. At the **sender**, the buffer has **three types of chambers**. The white section contains empty chambers that can be filled by the sending process (producer). **The colored area holds bytes that have been sent but not yet acknowledged**. The 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 (piggy-backing), 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.

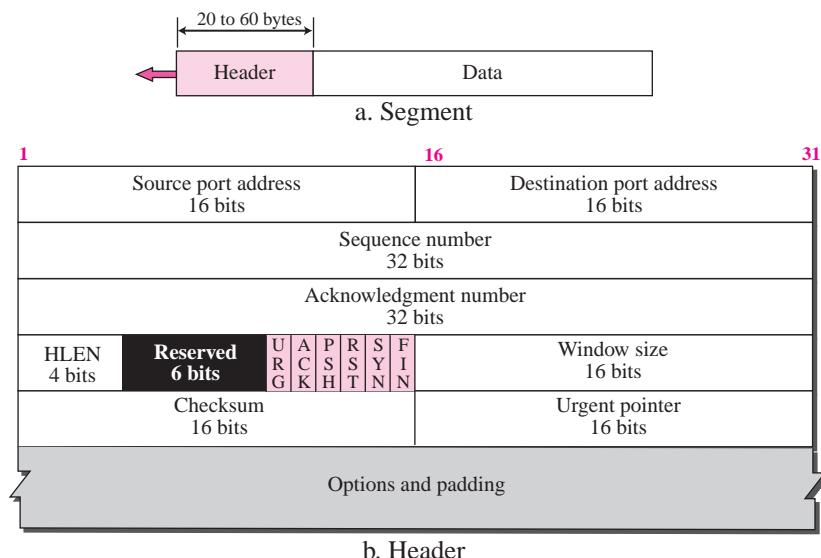
15.3 SEGMENT

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.

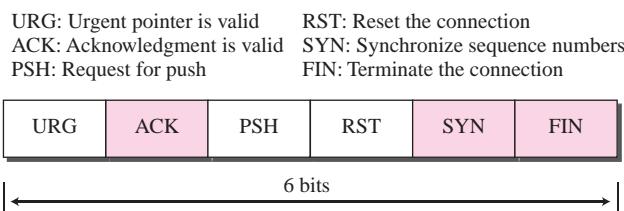
Figure 15.5 TCP segment format



- **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.

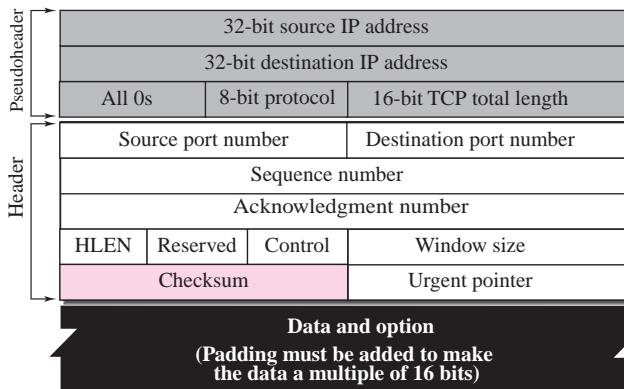
Figure 15.6 Control field



- ❑ **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*



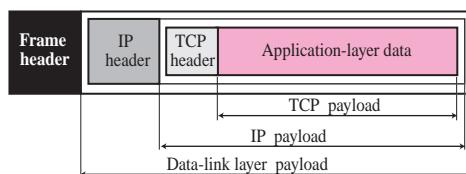
The use of the checksum in TCP is mandatory.

- ❑ **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.

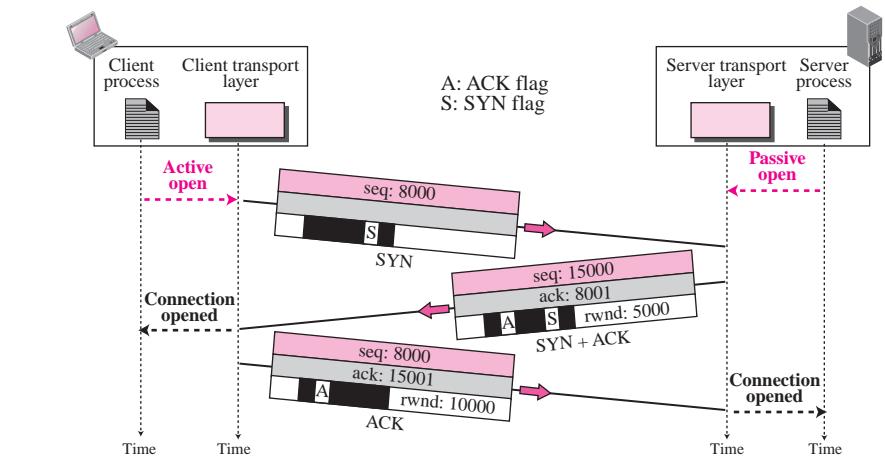
The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This request is called a **passive open**. Although the server TCP is ready to accept a connection from any machine in the world, it cannot make the connection itself.

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

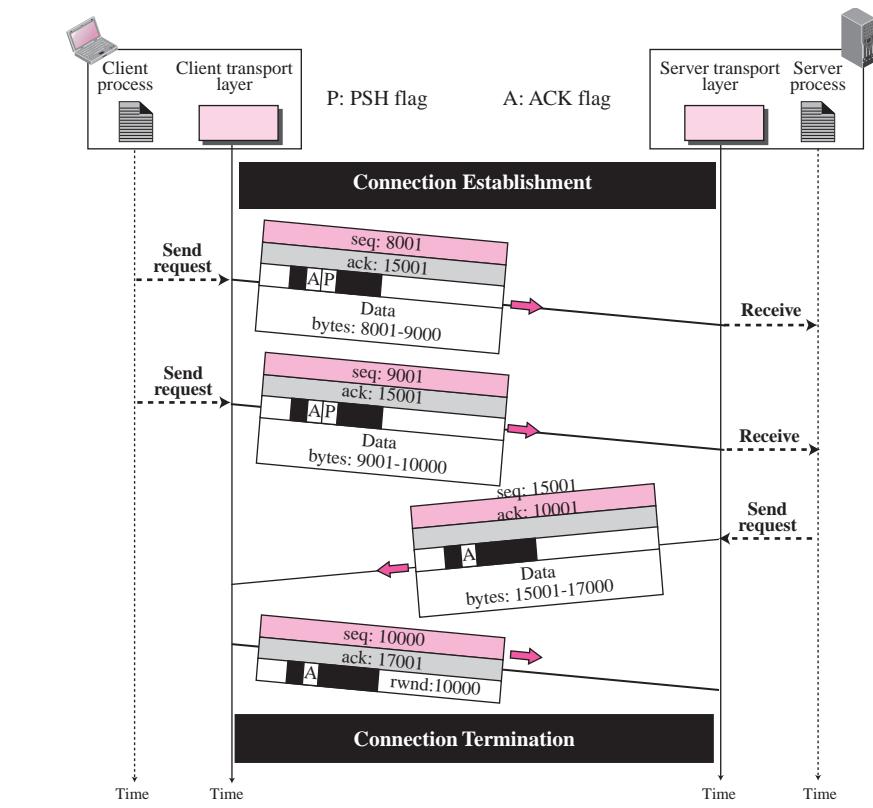
The connection establishment procedure in TCP is susceptible to a serious security problem called **SYN flooding attack**. This happens when one or more malicious attackers send a large number of SYN segments to a server pretending that each of them is coming from a different client by faking the source 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.

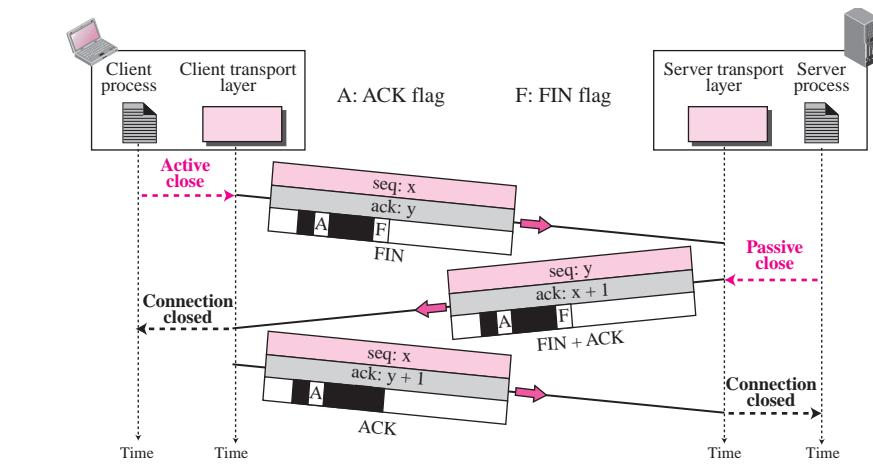
Three-Way Handshaking

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



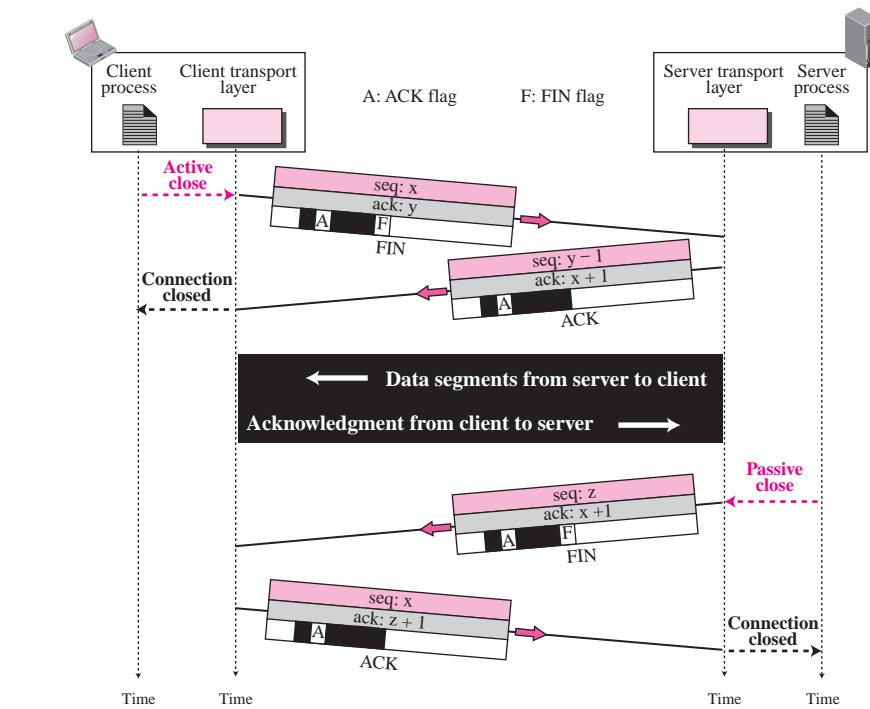
2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN+ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction. This segment can also 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.

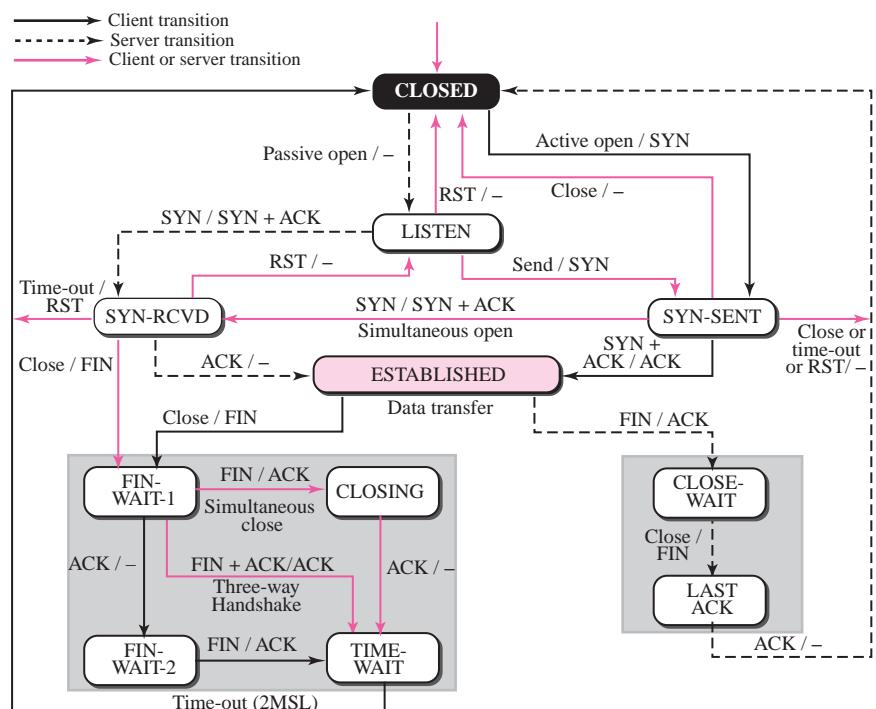
Terminating an Idle Connection

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 ESTABLISHED 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

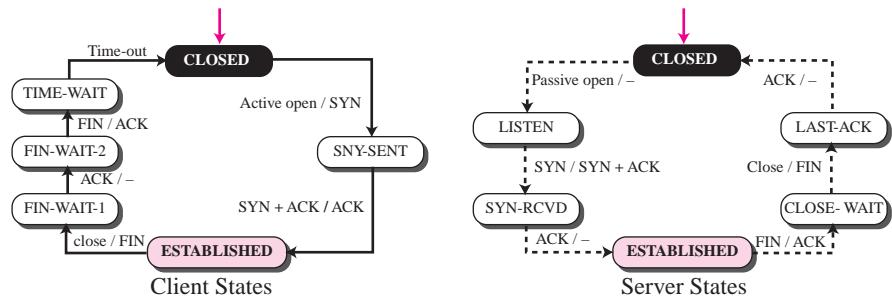
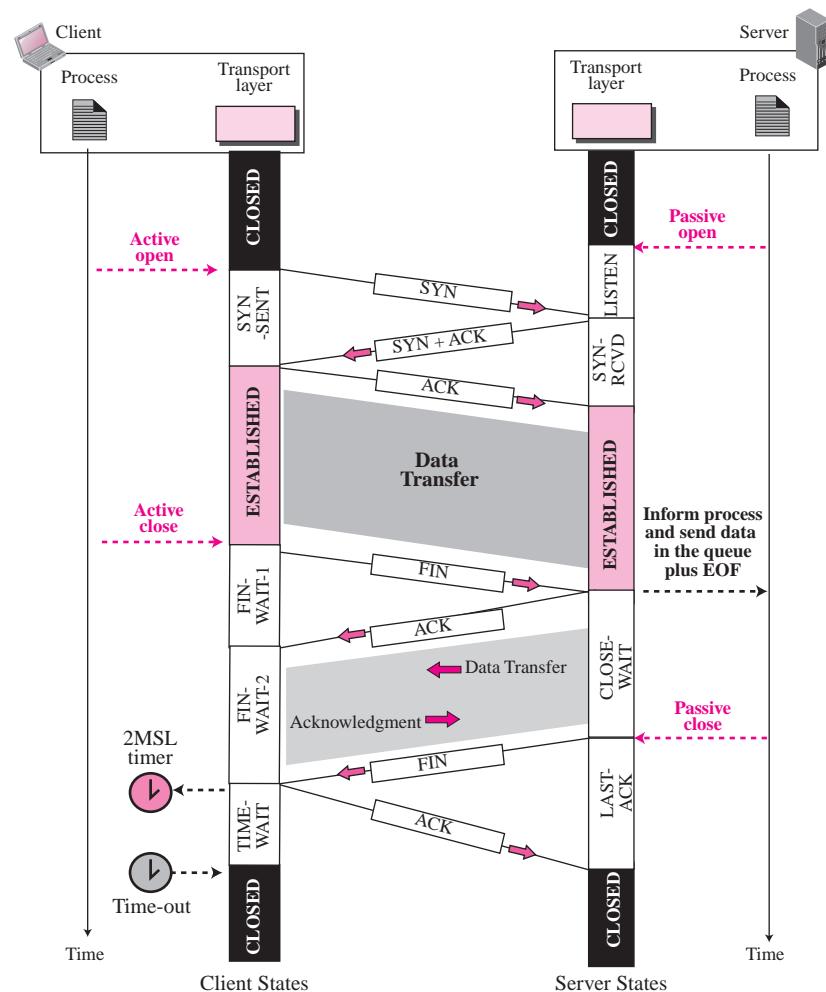
To understand the TCP state machines and the transition diagrams, we go through some scenarios in this section.

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

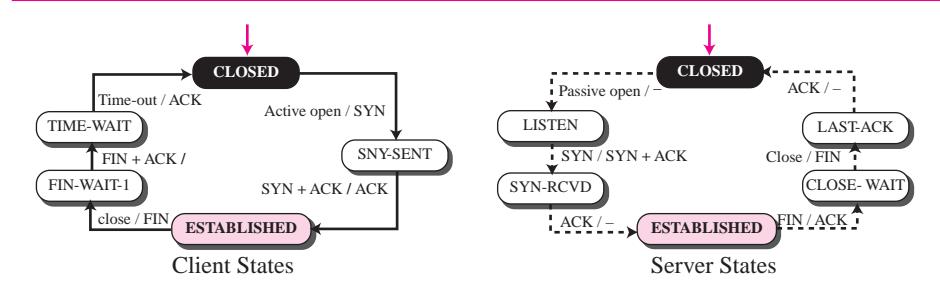
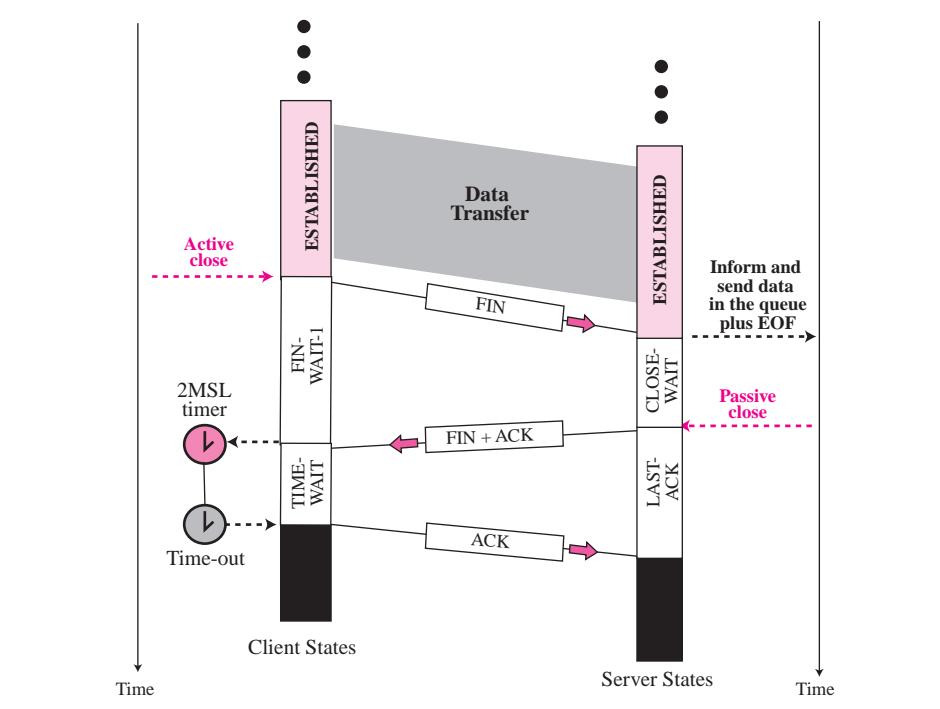


Figure 15.17 shows the same scenario with states over the time line. The connection establishment phase is the same as the one in the previous scenario; we show only the connection termination phase.

Figure 15.17 Time-line diagram for a common scenario

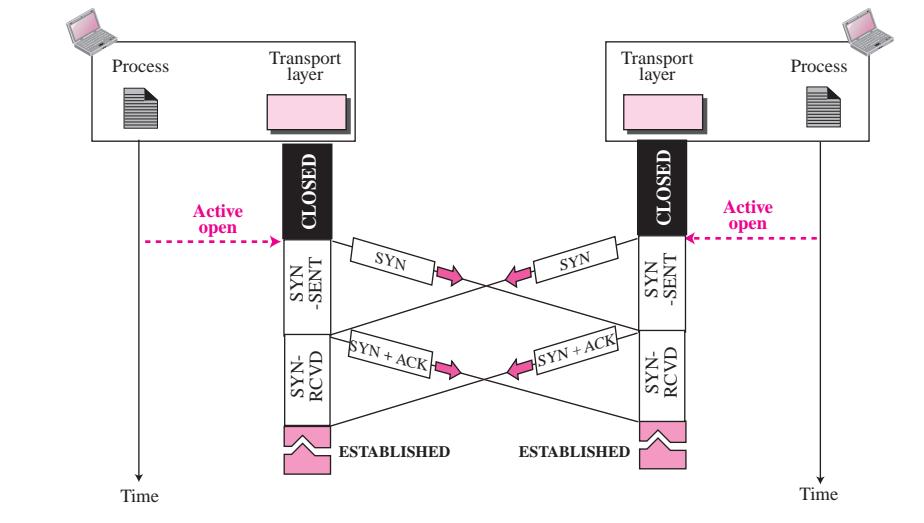


The figure shows that the client issues a *close* after the data transfer phase. The client TCP sends a FIN segment and goes to **FIN-WAIT-1** state. The server TCP, upon receiving the FIN segment, sends all queued data to the server with a virtual EOF marker, which means that the connection must be 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.

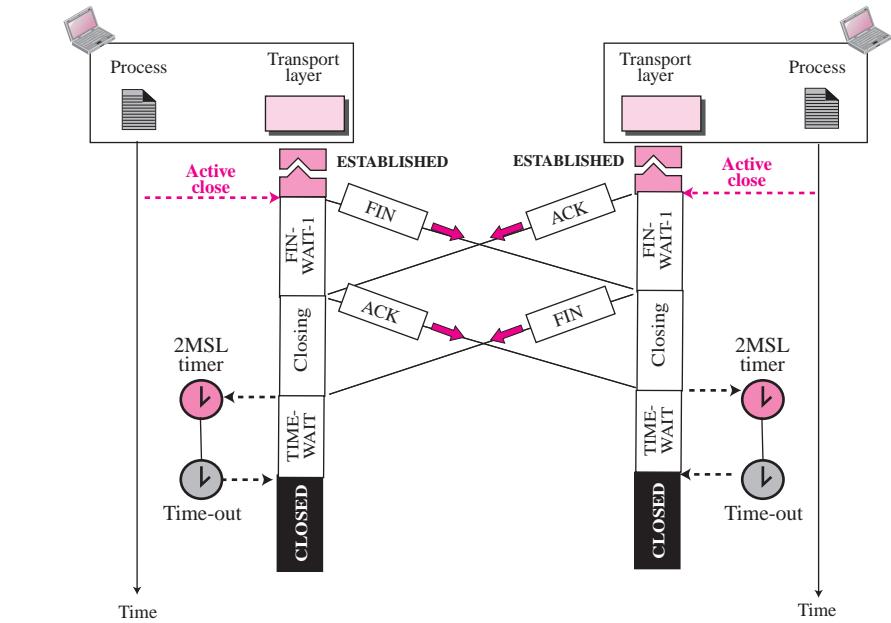
Figure 15.18 Simultaneous open



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.

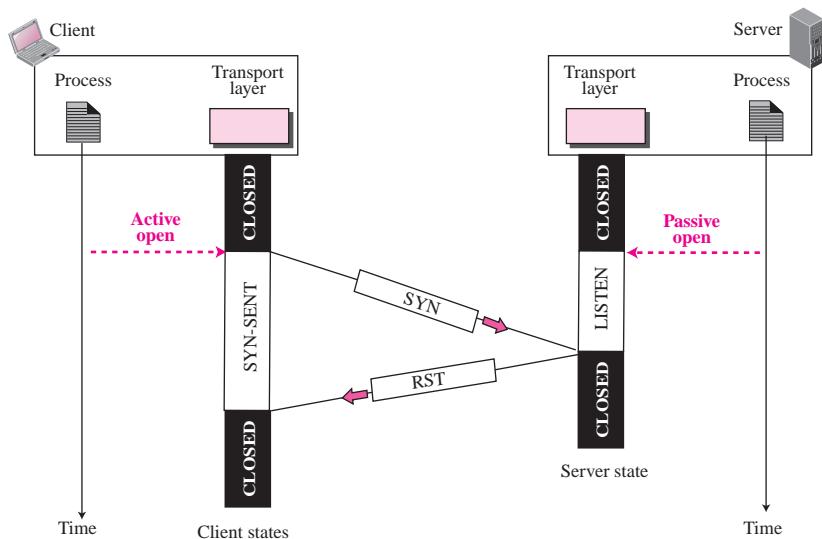
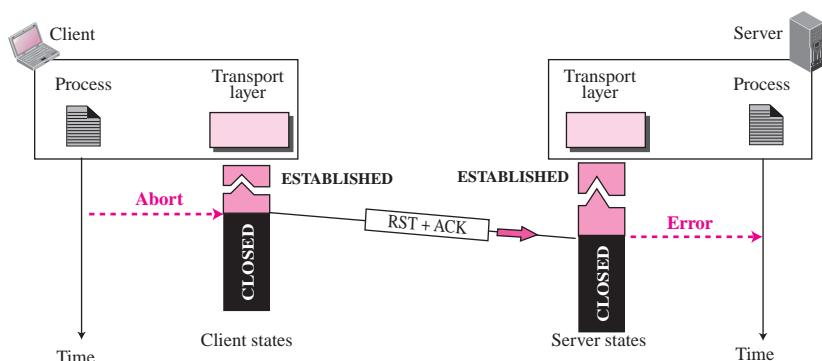
Figure 15.20 Denying a connection***Aborting a Connection***

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.

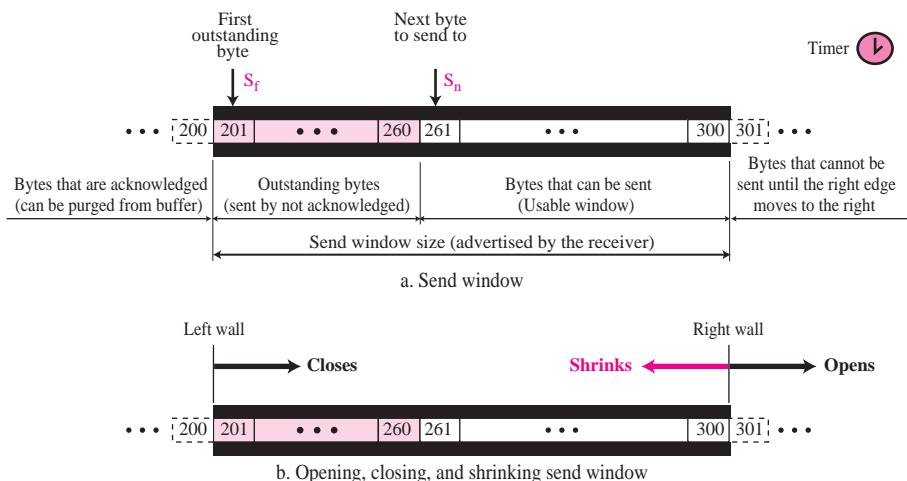
15.6 WINDOWS IN TCP

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 packets, but the window in the TCP numbers bytes. Although actual

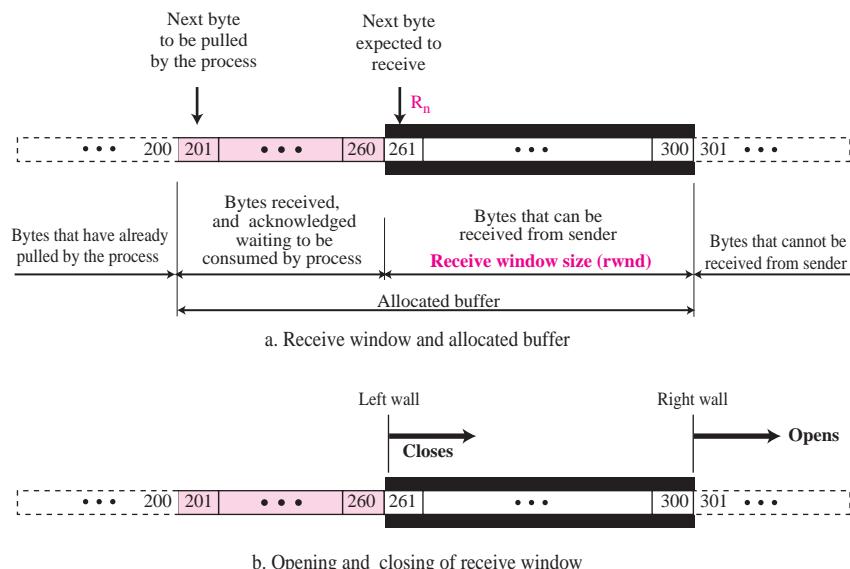
transmission in TCP occurs segment by segment, the variables that control the window are expressed in bytes.

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:

$$\text{rwnd} = \text{buffer size} - \text{number of waiting bytes to be pulled}$$

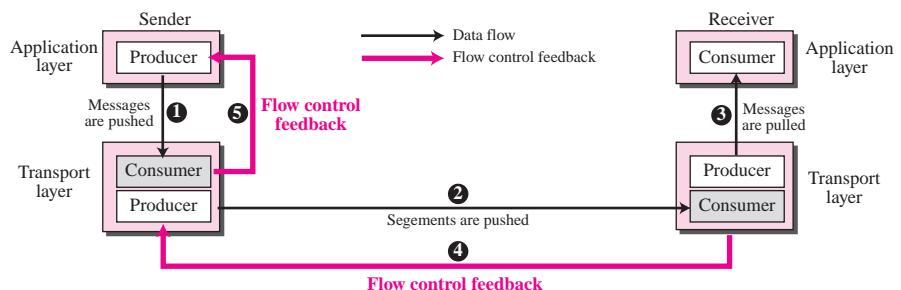
2. The second difference is the way acknowledgments are used in the TCP protocol. Remember that an acknowledgement in SR is selective, defining the uncorrupted packets that have been received. The major acknowledgment mechanism in TCP is a cumulative acknowledgment announcing the next expected byte to 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 wall 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.

P A R T

4

Application Layer

- | | | |
|------------|--|-----|
| Chapter 17 | Introduction to the Application Layer | 542 |
| Chapter 18 | Host Configuration: DHCP | 568 |
| Chapter 19 | Domain Name System (DNS) | 582 |
| Chapter 20 | Remote Login: TELNET and SSH | 610 |
| Chapter 21 | File Transfer: FTP and TFTP | 630 |
| Chapter 22 | World Wide Web and HTTP | 656 |
| Chapter 23 | Electronic Mail: SMTP, POP, IMAP, and MIME | 680 |
| Chapter 24 | Network Management: SNMP | 706 |
| Chapter 25 | Multimedia | 728 |

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.

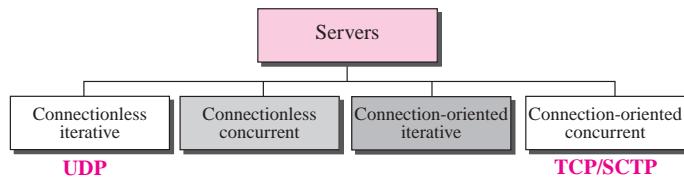
Concurrency in Clients

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.

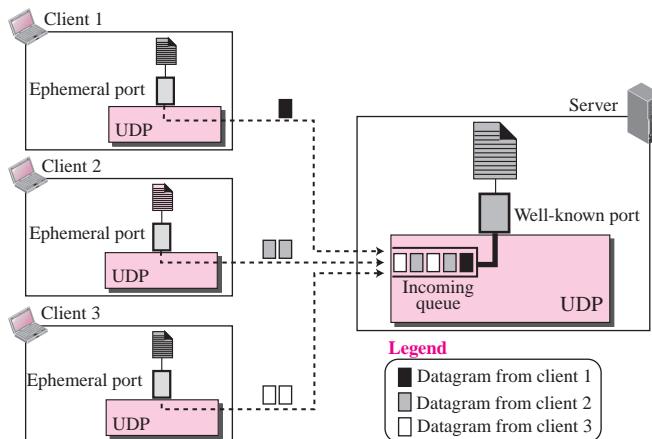
The servers use either UDP, a connectionless transport layer protocol, or TCP/SCTP, a connection-oriented transport layer protocol. Server operation, therefore, depends on two factors: the transport layer protocol and the service method. Theoretically we can have four types of servers: connectionless 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

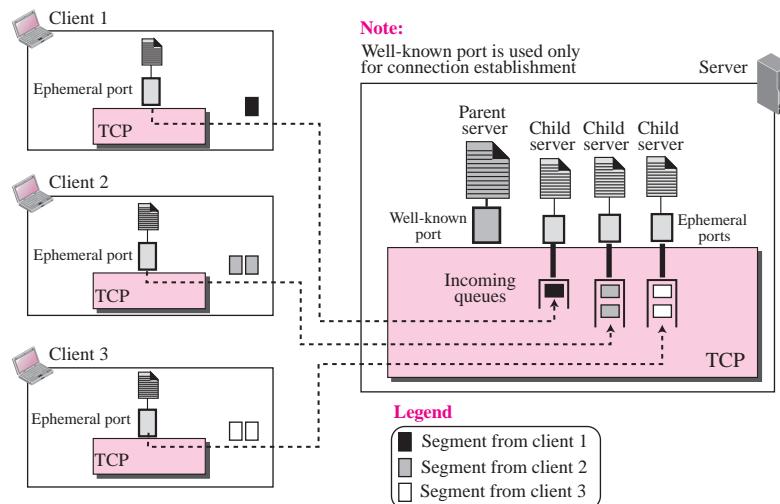
Connection-Oriented Concurrent Server

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

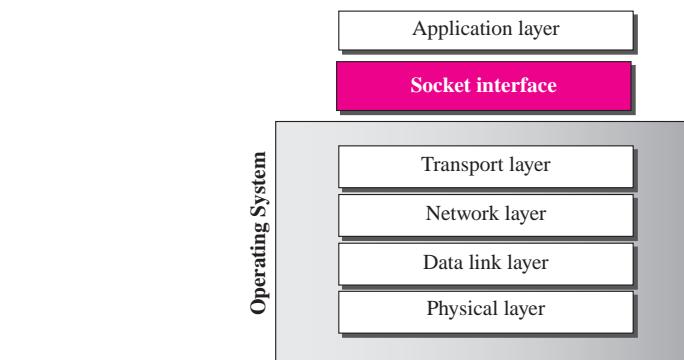
How can a client process communicate with a server process? A computer program is a set of predefined instructions that tells the computer what to do. A computer program has a set of instructions for mathematical operations, another set of instructions for string manipulation, still another set of instructions 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



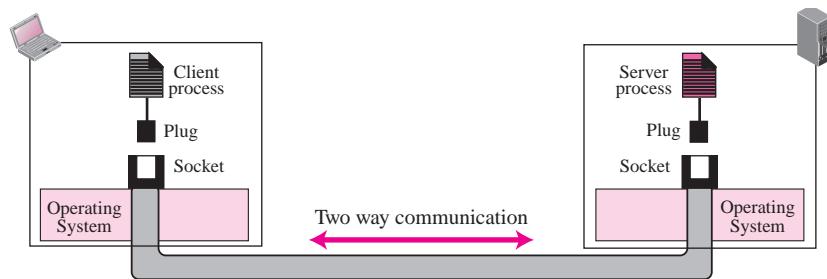
The socket interface, as a set of instructions, is located between the operating system and the application programs. To access the services provided by the TCP/IP protocol suite, an application needs to use the instructions defined in the socket interface.

Example 17.1

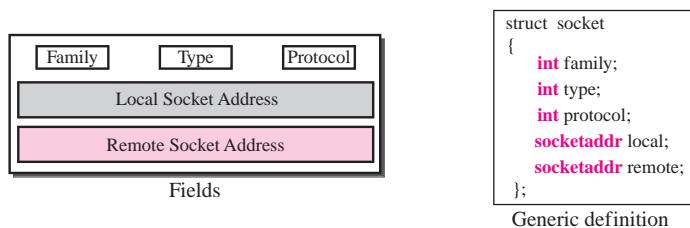
Most of the programming languages have a *file interface*, a set of instructions that allow the programmer to open a file, read from the file, write to the file, perform other operations on the file, and finally close the file. When a program needs to open the file, it uses the name of the file as it 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.

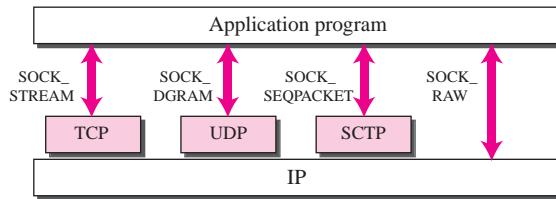
Figure 17.5 Concept of sockets**Data Structure**

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.

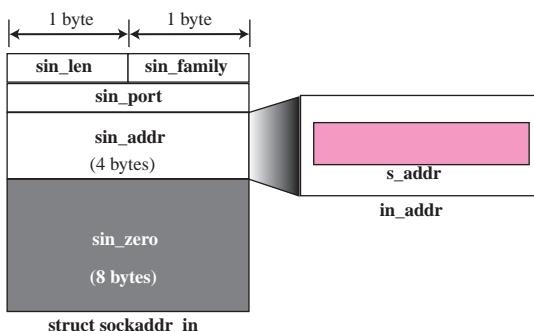
Figure 17.6 Socket data structure

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 sockaddr* 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 sockaddr* 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 sockaddr* 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

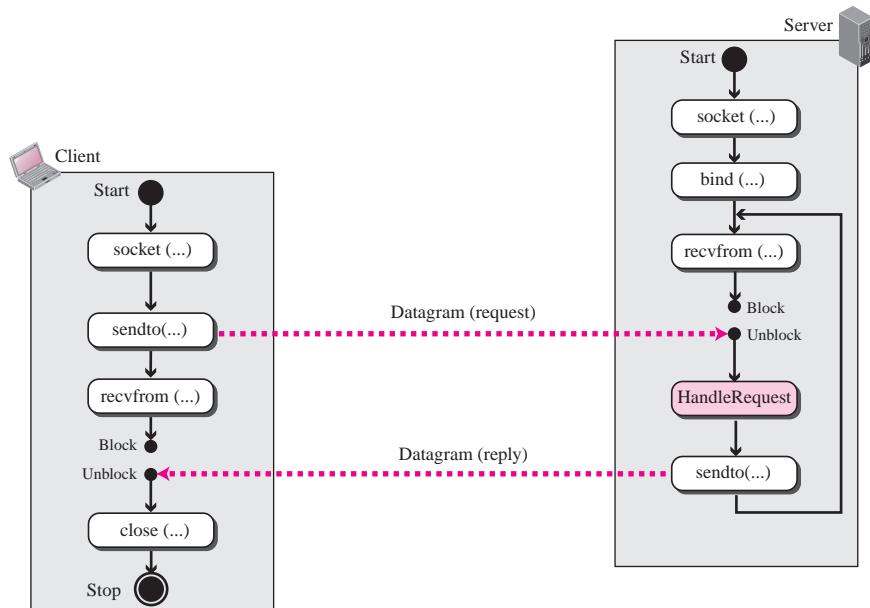
To use previously described functions, we need a set of header files. We define this header file in a separate file, which we name "headerFiles.h". We include this file in our programs to avoid including long lists of header files. Not all of these header files may be needed in all programs, but it is 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/inet.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

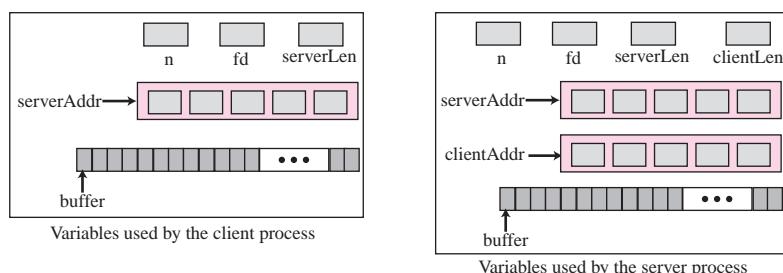


Table 17.1 shows the program for the echo server. We have eliminated many details and error-handling to simplify the program. We ask the reader to provide more details in exercises.

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 (&serverAddr, 0, sizeof(serverAddr));
15     serverAddr.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.

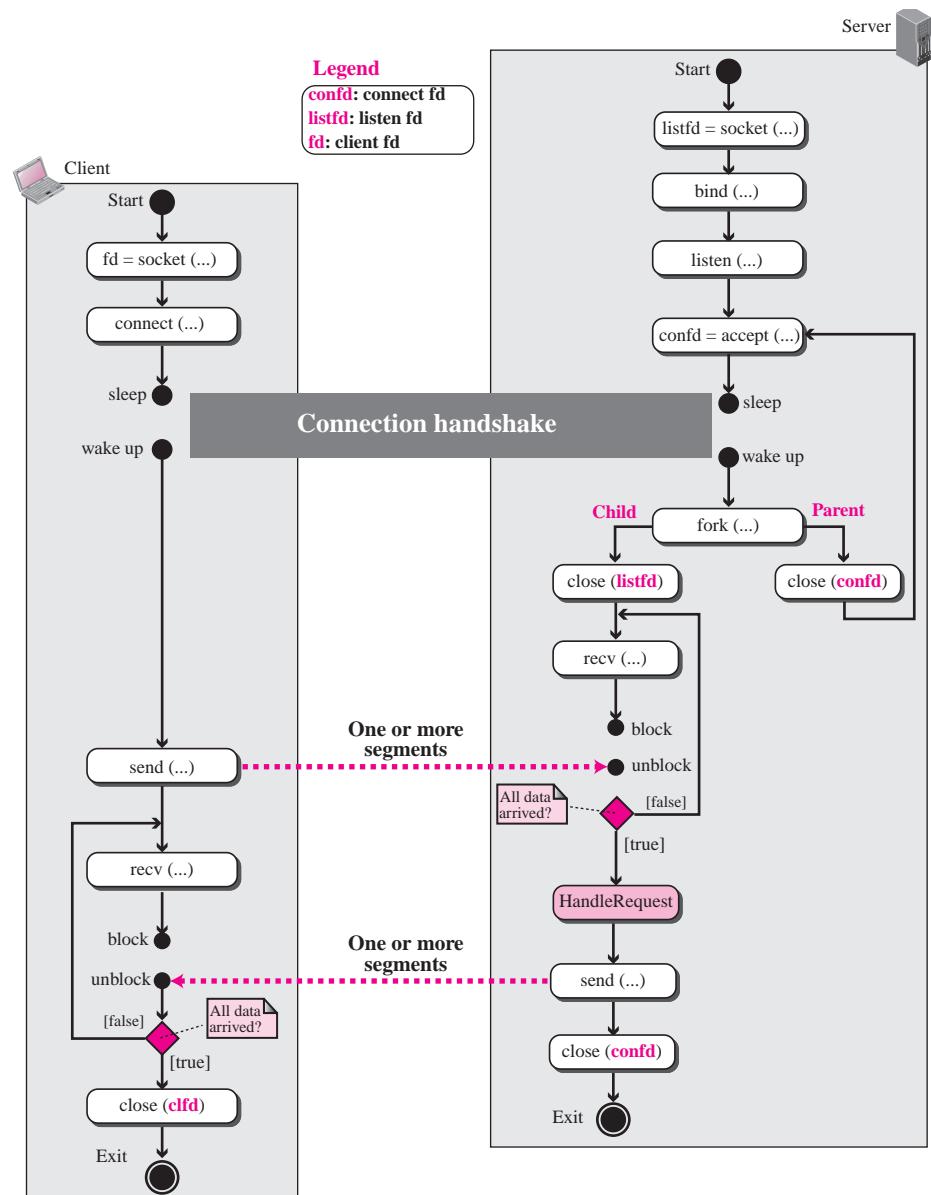
Communication Using TCP

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

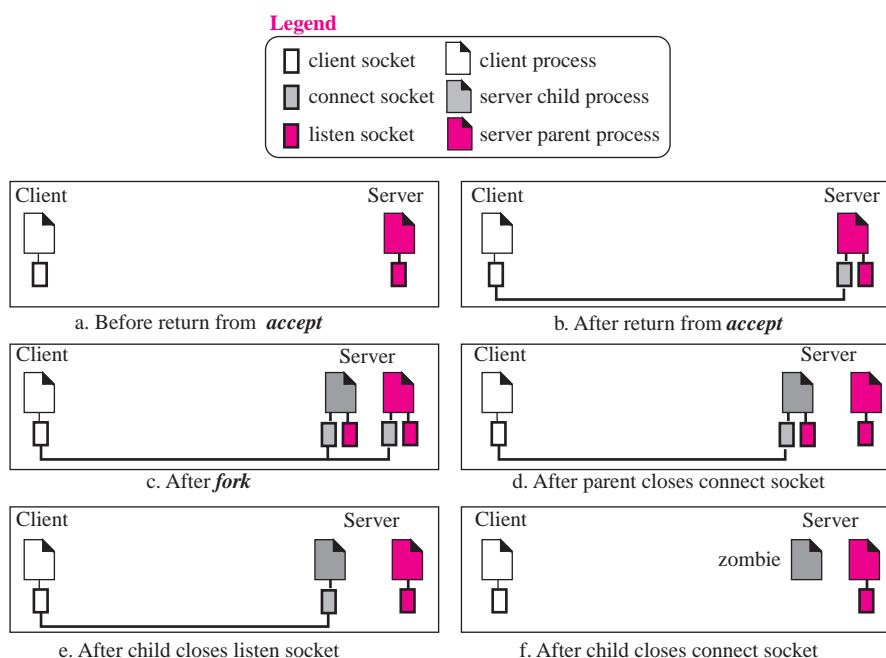


function; it is blocked until a segment arrives. The child process uses a loop and calls the *recv* function repeatedly until it receives all segments sent by the client. The child process then gives the whole data to a function (we call it *handleRequest*), to handle the request and return the result. 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 Variables used Example 17.3.

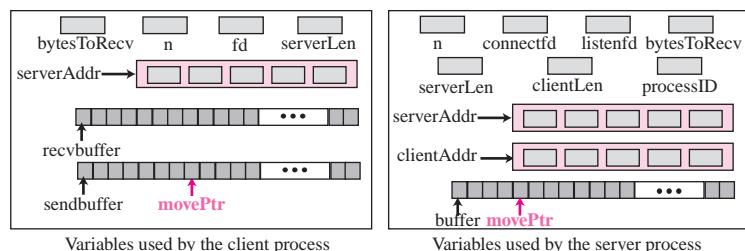


Table 17.3 shows the server program for an echo server that uses the services of TCP. We have eliminated many details and left them for the books on the network programming. We want just to show a global picture of the program.

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
18     // Create listen socket
19     listensd = socket (PF_INET, SOCK_STREAM, 0);
20     // Bind listen socket to the local address and port
21     memset (&serverAddr, 0, sizeof (serverAddr));
22     serverAddr.sin_family = AF_INET;
23     serverAddr.sin_addr.s_addr = htonl (INADDR_ANY);
24     serverAddr.sin_port = htons (7); // We assume port 7
25     bind (listensd, &serverAddr, sizeof (serverAddr));
26
27     // Listen to connection requests
28     listen (listensd, 5);
29
30     // Handle the connection
31     for ( ; ; )           // Run forever
32     {
33         connectsd = accept (listensd, &clientAddr, &clAddrLen);
34         processID = fork ();
35         if (processID == 0)           // Child process
36         {
37             close (listensd);
38             bytesToRecv = 256;
39             movePtr = buffer;
40             while ( (n = recv (connectfd, movePtr, bytesToRecv, 0)) > 0)
41             {
42                 movePtr = movePtr + n;
43                 bytesToRecv = movePtr - n;
44             } // 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.

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.

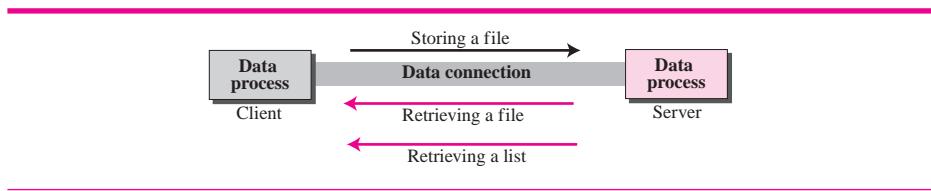
Table 21.7 Responses (continued)

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

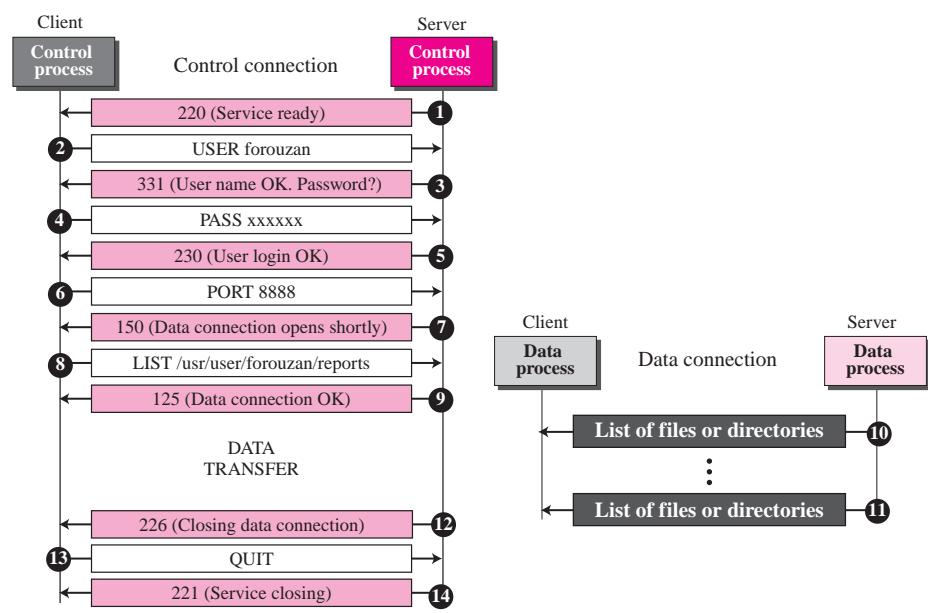
File transfer occurs over the data connection under the control of the commands sent over the control connection. However, we should remember that file transfer in FTP means one of three things (see Figure 21.7).

- 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.

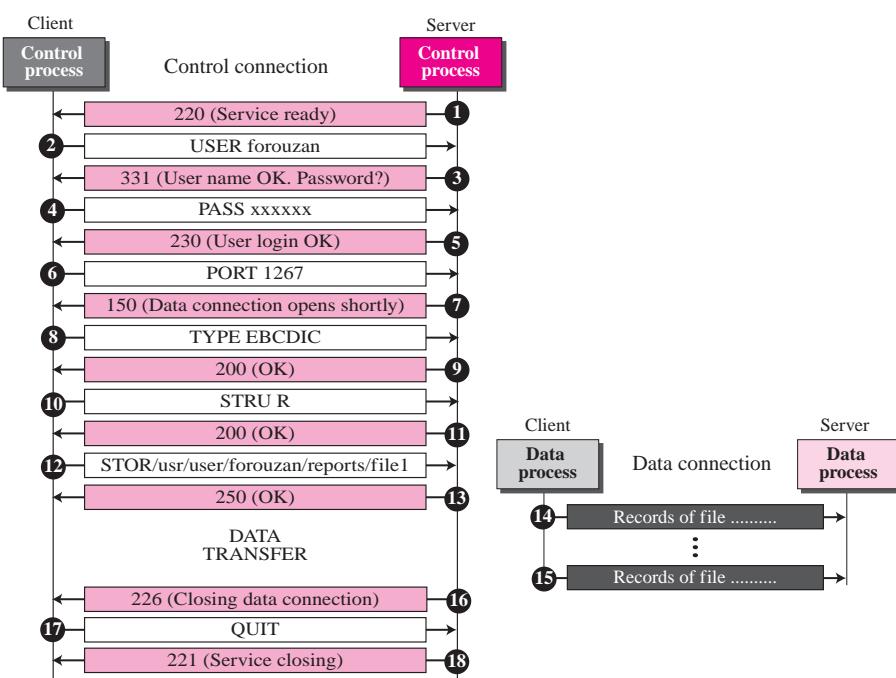
drwxr-xr-x  2  3027  411  4096 Sep 24  2002  business
drwxr-xr-x  2  3027  411  4096 Sep 24  2002  personal
drwxr-xr-x  2  3027  411  4096 Sep 24  2002  school

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

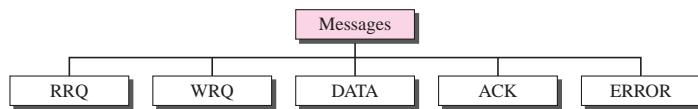
requires only basic IP and UDP. However, there is no security for TFTP. TFTP can read or write a file for the client. *Reading* means copying a file from the server site to the client site. *Writing* means copying a file from the client site to the server site.

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

OpCode = 1 2 bytes	File name Variable	All 0s 1 byte	Mode Variable	All 0s 1 byte
-----------------------	-----------------------	------------------	------------------	------------------

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

OpCode = 2 2 bytes	File name Variable	All 0s 1 byte	Mode Variable	All 0s 1 byte
-----------------------	-----------------------	------------------	------------------	------------------

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

OpCode = 3 2 bytes	Block number 2 bytes	Data 0–512 bytes
-----------------------	-------------------------	---------------------

- ❑ **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

OpCode = 4 2 bytes	Block number 2 bytes
-----------------------	-------------------------

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*

OpCode = 5	Error number	Error data	All 0s
2 bytes	2 bytes	Variable	1 byte

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.

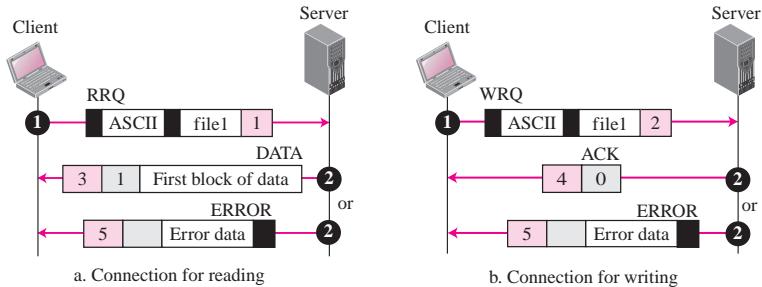
Connection

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.

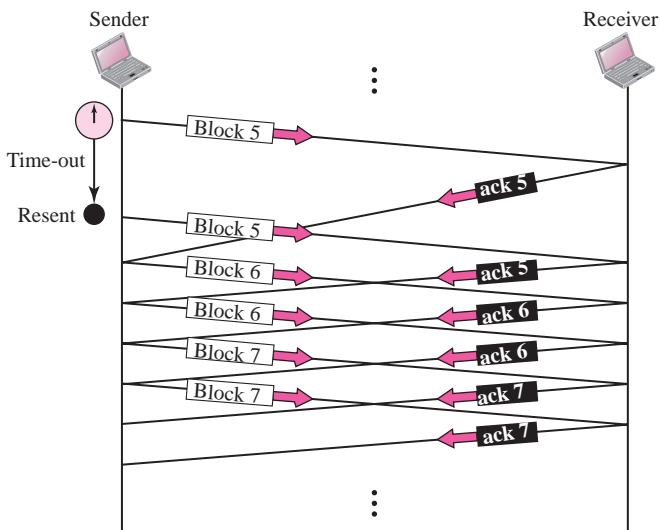
Duplicate Message Duplication of blocks can be detected by the receiver through block number. If a block is duplicated, it is simply discarded by the receiver.

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



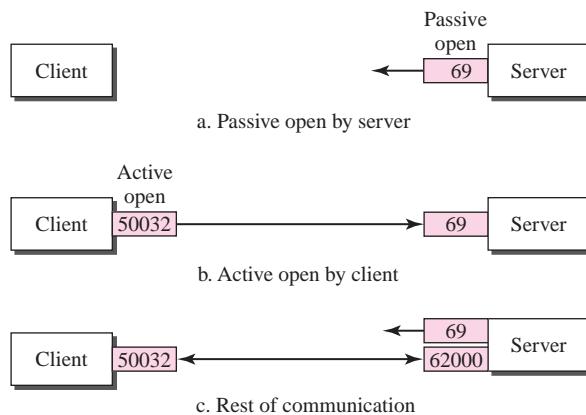
UDP Ports

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

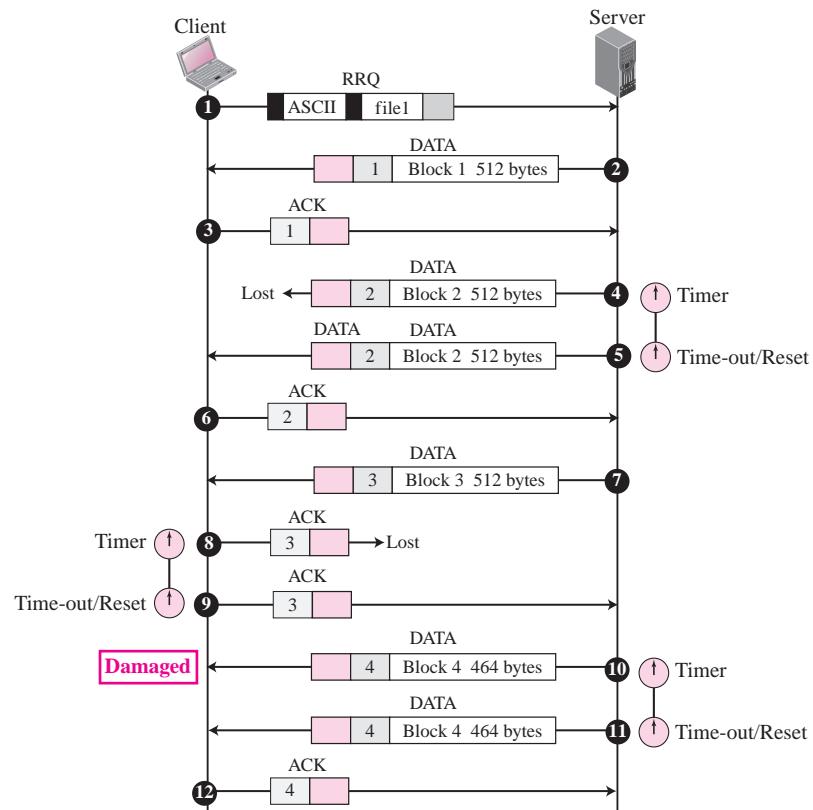
3. The server actively opens a connection using a new ephemeral port for the source port and uses the ephemeral port received from the client as the destination port. It sends the DATA or ACK or ERROR message using these ports. This frees the well-known port (69) for use by other clients. When the client 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

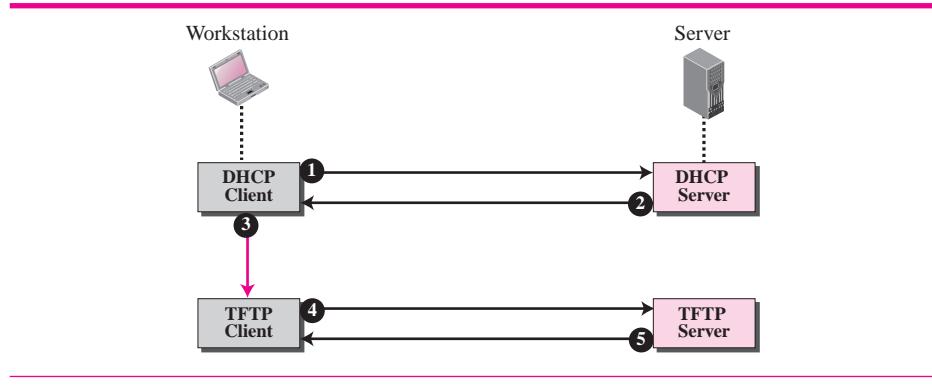
One important point about TFTP is that there is no provision for security: There is no user identification or password. Today, however, precautions must be taken to prevent hackers from accessing files. One security measure is to limit the access of TFTP to noncritical files. One way to achieve minimal 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

data connection

ASCII file

EBCDIC file

block mode

file structure

compressed mode

File Transfer Protocol (FTP)

connection establishment

flow control

control connection

image file

reading	stream mode
record structure	Trivial File Transfer Protocol (TFTP)
sorcerer's apprentice bug	writing
sftp	

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.

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.

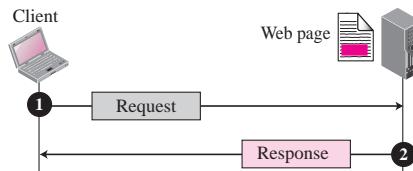
22.1 ARCHITECTURE

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.

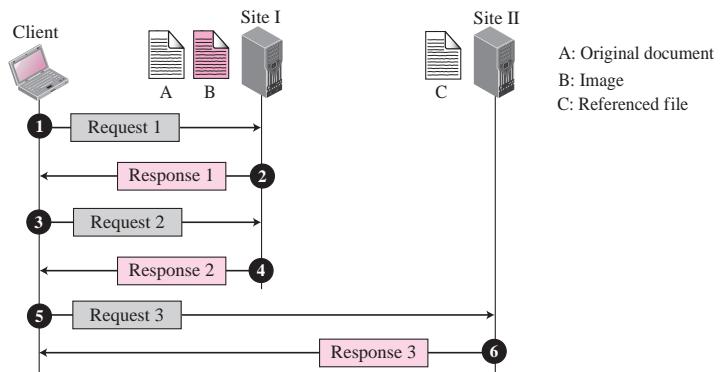
Figure 22.1 Example 22.1



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**

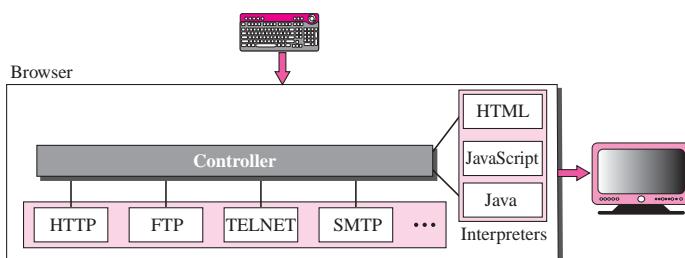
A very important point we need to remember is that file A, file B, and file C in Example 22.2 are independent Web pages, each with independent names and addresses. Although references to file B or C are included in file A, it does not mean that each of these files cannot be retrieved independently. A 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.

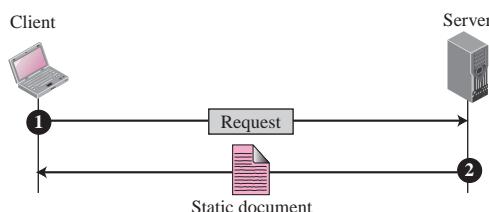
22.2 WEB DOCUMENTS

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.

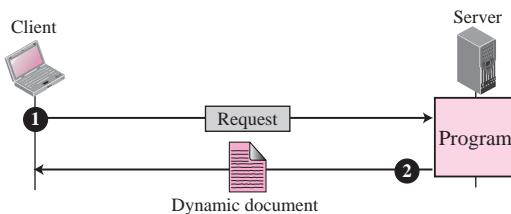
Common Gateway Interface (CGI)

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.

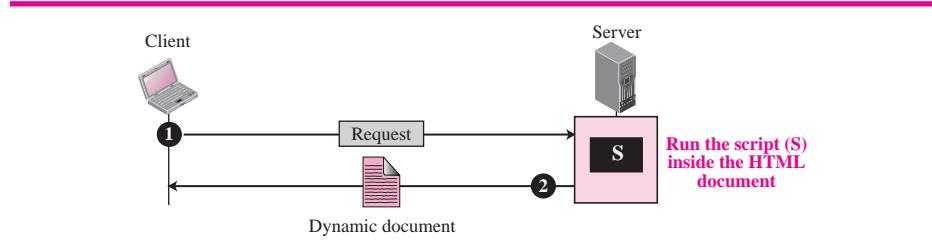
Output The whole idea of CGI is to execute a CGI program at the server site and send the output to the client (browser). The output is usually plain text or a text with HTML structures; however, the output can be a variety of other things. It can be graphics or binary data, a status code, instructions 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

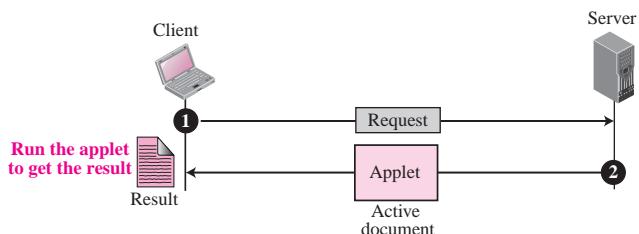
For many applications, we need a program or a script to be run at the client site. These are called **active documents**. For example, suppose we want to run a program that creates animated graphics on the screen or a program that interacts with the user. The program definitely needs to be run at the 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.

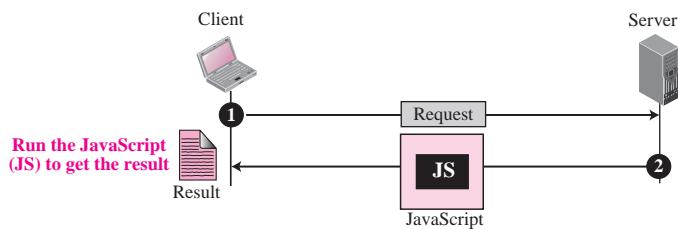
An **applet** is a program written in Java on the server. It is compiled and ready to be run. The document is in bytecode (binary) format. The client process (browser) creates an instance of this applet and runs it. A Java applet can be run by the browser in two ways. In the first method, the browser 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,