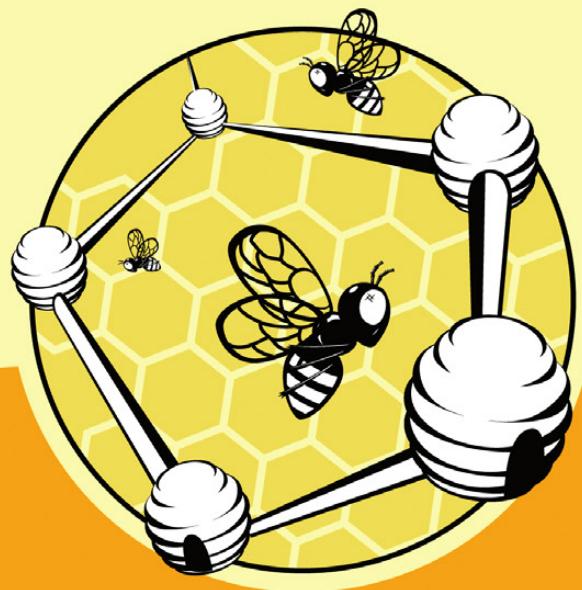


COVERS IPv4 AND IPv6

The

# TCP/IP GUIDE

A COMPREHENSIVE, ILLUSTRATED  
INTERNET PROTOCOLS REFERENCE



Charles M. Kozierok



## **What Readers Are Saying about *The TCP/IP Guide***

“Whenever I need information or clarification on any TCP/IP or OSI related topic, *The TCP/IP Guide* is the first place I go.”

—SHAWN BRIDGLAL, COMPUTER ENGINEERING TECHNOLOGIST

“ . . . an extremely readable and lucid document on a very involved subject. *The TCP/IP Guide* is an immense piece of work, but it does the best job I’ve seen of making a complicated subject understandable.”

—PETER BICE, CONSULTING SOFTWARE ENGINEER

“An absolute must-have for the serious student of networking, and those who just want a greater understanding of the technical world around them. Its level of detail is excellent, yet its simple explanations and analogies of even the most complex subjects make this work unparalleled. Even for the experienced student its coverage of the basics can make things so much clearer.”

—IAIN DENTON, NETWORKING STUDENT

“Written in Charles M. Kozierok’s familiar down-to-earth style, *The TCP/IP Guide* is at once easy enough to understand that it should be an excellent starting place for beginners to learn networking—and at the same time, thorough and in-depth enough to make it a must-have reference for advanced users and professionals.”

—BILLY GRIFFIS, COMPUTER TECHNICIAN

“In *The TCP/IP Guide*, Charles M. Kozierok thoroughly explains the details of many complex networking processes in a straightforward, orderly manner. He is an excellent teacher—his lucid understanding of his subject matter and his awareness of his audience are evident in every explanation of the many TCP/IP topics he addresses. *The TCP/IP Guide* is a valuable resource for anyone wanting to thoroughly understand the details of how TCP/IP networking works.”

—DAVID WILLIAMS, TEACHER

“Everyone who is starting off their IT career or wanting to increase their knowledge of TCP/IP would benefit enormously by buying this book.”

—KIRK PASCOE, INFORMATION SYSTEMS ENGINEER



# **THE TCP/IP GUIDE**

**A Comprehensive, Illustrated  
Internet Protocols Reference**

**by Charles M. Kozierok**



San Francisco

**THE TCP/IP GUIDE.** Copyright © 2005 by Charles M. Kozierok.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

12 11 10                5 6 7 8 9 10

Printed in the United States of America

ISBN-10: 1-59327-047-X

ISBN-13: 978-1-59327-047-6

Publisher: William Pollock

Production Manager: Susan Berge

Cover and Interior Design: Octopod Studios

Developmental Editor: William Pollock

Technical Reviewers: Fernando Gont, Barry Margolin

Copyeditors: Marilyn Smith, Lisa Theobald, Mark Nigara

Compositor: Riley Hoffman

Proofreader: Stephanie Provines

For information on book distributors or translations, please contact No Starch Press, Inc. directly:

No Starch Press, Inc.

38 Ringold Street, San Francisco, CA 94103

phone: 415.863.9900; fax: 415.863.9950; info@nostarch.com; http://www.nostarch.com

*Library of Congress Cataloging-in-Publication Data*

Kozierok, Charles.

The TCP/IP guide : a comprehensive, illustrated internet protocols reference / Charles Kozierok.

p. cm.

Includes index.

ISBN 1-59327-047-X

1. TCP/IP (Computer network protocol) I. Title.

TK5105.585.K69 2005

004.6'2--dc22

2004008863

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

## **DEDICATION**

This book is dedicated to my family: my wife, Robyn, and my three sons, Ryan, Matthew, and Evan.

I suppose that it's a cliché to dedicate a book to your wife. If so, it's for a very good reason. Who plays a more important role in the life of an author than his or her spouse? Robyn is my partner—the person who is always there and the one who shares my life in so many ways. The expression about a great woman being behind every good man is true, yet my wife is deserving of recognition for reasons that go far beyond the usual one of being "supportive."

She agreed to take on a regular working position to make it possible for me to spend time on a very long project with an uncertain payoff. She took on most of the tasks of taking our children to school and dealing with their needs, to give me time to write. She also gracefully agreed to "do without" many things that many other wives would not have been too happy about forgoing.

But most of all, she deserves a world of credit for putting up with *me*. For constantly reassuring me that it was okay that I was spending years on a project that might not be successful. For listening to me talk for countless hours, and for giving her opinions on many portions of my writing, all on a subject that really doesn't interest her. And most important, for encouraging me when I felt this was a waste of time, and even kicking me in the butt when I felt like giving up. Without Robyn, this book simply would not exist. Thanks, R.

My three boys deserve credit for similar reasons, but to a lesser extent. They have had to put up with my constantly sitting at the computer, trying to tune them out so I could concentrate; my too-frequent grouchy moods; and my reluctance to spend time with them when I had work on my plate. I am sure there were many times that they wished I just had a regular "day job."

Ryan, my big boy, has been very patient in waiting for me to finish this project so we can resume several activities that we used to engage in regularly. Matthew, my fun-loving and rambunctious middle son, has also had to deal with me not being able to spend as much time as I would have liked with him. And little Evan has had a father working on a long-term project for his entire life! All three of my boys have been very understanding and provided me with much-needed joy and laughter at times when I needed them most.

## BRIEF CONTENTS

Contents in Detail.....	xiii
List of Figures.....	.xlv
List of Tables.....	lv
Acknowledgments .....	lxv
About the Author.....	lxvii
Introduction .....	lxix

## SECTION I: TCP/IP OVERVIEW AND BACKGROUND INFORMATION

### PART I-1: NETWORKING FUNDAMENTALS

Chapter 1: Networking Introduction, Characteristics, and Types .....	5
Chapter 2: Network Performance Issues and Concepts .....	31
Chapter 3: Network Standards and Standards Organizations .....	45
Chapter 4: A Review of Data Representation and the Mathematics of Computing .....	61

### PART I-2: THE OPEN SYSTEMS INTERCONNECTION (OSI) REFERENCE MODEL

Chapter 5: General OSI Reference Model Issues and Concepts .....	81
Chapter 6: OSI Reference Model Layers .....	101
Chapter 7: OSI Reference Model Summary .....	113

### PART I-3: TCP/IP PROTOCOL SUITE AND ARCHITECTURE

Chapter 8: TCP/IP Protocol Suite and Architecture .....	121
---------------------------------------------------------	-----

## **SECTION II: TCP/IP LOWER-LAYER CORE PROTOCOLS**

### **PART II-1: TCP/IP NETWORK INTERFACE LAYER PROTOCOLS**

Chapter 9: TCP/IP Serial Line Internet Protocol (SLIP) and Point-to-Point Protocol (PPP) Overview and Fundamentals.....	139
Chapter 10: PPP Core Protocols: Link Control, Network Control, and Authentication .....	155
Chapter 11: PPP Feature Protocols.....	167
Chapter 12: PPP Protocol Frame Formats.....	181

### **PART II-2: TCP/IP NETWORK INTERFACE/INTERNET LAYER CONNECTION PROTOCOLS**

Chapter 13: Address Resolution and the TCP/IP Address Resolution Protocol (ARP) .....	203
Chapter 14: Reverse Address Resolution and the TCP/IP Reverse Address Resolution Protocol (RARP).....	227

### **PART II-3: INTERNET PROTOCOL VERSION 4 (IP/IPv4)**

Chapter 15: Internet Protocol Versions, Concepts, and Overview .....	235
Chapter 16: IPv4 Addressing Concepts and Issues .....	241
Chapter 17: Classful (Conventional) Addressing .....	255
Chapter 18: IP Subnet Addressing (Subnetting) Concepts .....	273
Chapter 19: IP Subnetting: Practical Subnet Design and Address Determination Example.....	297
Chapter 20: IP Classless Addressing—Classless Inter-Domain Routing (CIDR)/Supernetting .....	315
Chapter 21: Internet Protocol Datagram Encapsulation and Formatting .....	329
Chapter 22: IP Datagram Size, Fragmentation, and Reassembly .....	339
Chapter 23: IP Routing and Multicasting .....	351

### **PART II-4: INTERNET PROTOCOL VERSION 6 (IPV6)**

Chapter 24: IPv6 Overview, Changes, and Transition .....	365
Chapter 25: IPv6 Addressing.....	373
Chapter 26: IPv6 Datagram Encapsulation and Formatting.....	401
Chapter 27: IPv6 Datagram Size, Fragmentation, Reassembly, and Routing .....	415

## **PART II-5: IP-RELATED FEATURE PROTOCOLS**

Chapter 28: IP Network Address Translation (NAT) Protocol.....	425
Chapter 29: IP Security (IPsec) Protocols .....	449
Chapter 30: Internet Protocol Mobility Support (Mobile IP) .....	475

## **PART II-6: IP SUPPORT PROTOCOLS**

Chapter 31: ICMP Concepts and General Operation .....	507
Chapter 32: ICMPv4 Error Message Types and Formats .....	521
Chapter 33: ICMPv4 Informational Message Types and Formats.....	535
Chapter 34: ICMPv6 Error Message Types and Formats .....	547
Chapter 35: ICMPv6 Informational Message Types and Formats.....	557
Chapter 36: IPv6 Neighbor Discovery (ND) Protocol .....	575

## **PART II-7: TCP/IP ROUTING PROTOCOLS (GATEWAY PROTOCOLS)**

Chapter 37: Overview of Key Routing Protocol Concepts .....	591
Chapter 38: Routing Information Protocol (RIP, RIP-2, and RIPng).....	597
Chapter 39: Open Shortest Path First (OSPF) .....	625
Chapter 40: Border Gateway Protocol (BGP/BGP-4) .....	647
Chapter 41: Other Routing Protocols .....	677

## **PART II-8: TCP/IP TRANSPORT LAYER PROTOCOLS**

Chapter 42: Overview and Comparison of TCP and UDP .....	689
Chapter 43: TCP and UDP Addressing: Ports and Sockets .....	695
Chapter 44: TCP/IP User Datagram Protocol (UDP) .....	711
Chapter 45: TCP Overview, Functions, and Characteristics .....	719
Chapter 46: Transmission Control Protocol (TCP) Fundamentals and General Operation .....	727
Chapter 47: TCP Basic Operation: Connection Establishment, Management, and Termination.....	745
Chapter 48: TCP Message Formatting and Data Transfer .....	769
Chapter 49: TCP Reliability and Flow Control Features .....	793

## **SECTION III: TCP/IP APPLICATION LAYER PROTOCOLS**

### **PART III-1: NAME SYSTEMS AND TCP/IP NAME REGISTRATION AND NAME RESOLUTION**

Chapter 50: Name System Issues, Concepts, and Techniques .....	825
Chapter 51: TCP/IP Name Systems Overview and the Host Table Name System .....	841
Chapter 52: Domain Name System (DNS) Overview, Functions, and Characteristics.....	847
Chapter 53: DNS Name Space, Architecture, and Terminology.....	857
Chapter 54: DNS Name Registration, Public Administration, Zones, and Authorities .....	867
Chapter 55: DNS Name Server Concepts and Operation .....	887
Chapter 56: DNS Resolution Concepts and Resolver Operations.....	909
Chapter 57: DNS Messaging and Message, Resource Record, and Master File Formats .....	927

### **PART III-2: NETWORK FILE AND RESOURCE SHARING PROTOCOLS**

Chapter 58: Network File and Resource Sharing and the TCP/IP Network File System (NFS).....	953
------------------------------------------------------------------------------------------------	-----

### **PART III-3: HOST CONFIGURATION AND TCP/IP HOST CONFIGURATION PROTOCOLS**

Chapter 59: Host Configuration Concepts, Issues, and Motivation .....	973
Chapter 60: TCP/IP Bootstrap Protocol (BOOTP).....	977
Chapter 61: DHCP Overview and Address Allocation Concepts .....	997
Chapter 62: DHCP Configuration and Operation.....	1013
Chapter 63: DHCP Messaging, Message Types, and Formats .....	1035
Chapter 64: DHCP Client/Server Implementation, Features, and IPv6 Support.....	1053

### **PART III-4: TCP/IP NETWORK MANAGEMENT FRAMEWORK AND PROTOCOLS**

Chapter 65: TCP/IP Internet Standard Management Framework Overview.....	1069
Chapter 66: TCP/IP Structure of Management Information (SMI) and Management Information Bases (MIBs) .....	1083
Chapter 67: TCP/IP Simple Network Management Protocol (SNMP) Concepts and Operation .....	1099
Chapter 68: SNMP Protocol Messaging and Message Formats.....	1113

Chapter 69: TCP/IP Remote Network Monitoring (RMON) .....	1133
-----------------------------------------------------------	------

## **PART III-5: TCP/IP APPLICATION LAYER ADDRESSING AND APPLICATION CATEGORIES**

Chapter 70: TCP/IP Application Layer Addressing: Uniform Resource Identifiers, Locators, and Names (URLs, URLs, and URNs) .....	1139
---------------------------------------------------------------------------------------------------------------------------------	------

Chapter 71: File and Message Transfer Overview and Application Categories.....	1163
--------------------------------------------------------------------------------	------

## **PART III-6: TCP/IP GENERAL FILE TRANSFER PROTOCOLS**

Chapter 72: File Transfer Protocol (FTP) .....	1169
------------------------------------------------	------

Chapter 73: Trivial File Transfer Protocol (TFTP).....	1199
--------------------------------------------------------	------

## **PART III-7: TCP/IP ELECTRONIC MAIL SYSTEM: CONCEPTS AND PROTOCOLS**

Chapter 74: TCP/IP Electronic Mail System Overview and Concepts .....	1217
-----------------------------------------------------------------------	------

Chapter 75: TCP/IP Electronic Mail Addresses and Addressing .....	1225
-------------------------------------------------------------------	------

Chapter 76: TCP/IP Electronic Mail Message Formats and Message Processing: RFC 822 and MIME .....	1233
---------------------------------------------------------------------------------------------------	------

Chapter 77: TCP/IP Electronic Mail Delivery Protocol: The Simple Mail Transfer Protocol (SMTP) .....	1263
------------------------------------------------------------------------------------------------------	------

Chapter 78: TCP/IP Electronic Mail Access and Retrieval Protocols and Methods.....	1285
------------------------------------------------------------------------------------	------

## **PART III-8: TCP/IP WORLD WIDE WEB AND THE HYPERTEXT TRANSFER PROTOCOL (HTTP)**

Chapter 79: World Wide Web and Hypertext Overview and Concepts .....	1317
----------------------------------------------------------------------	------

Chapter 80: HTTP General Operation and Connections.....	1329
---------------------------------------------------------	------

Chapter 81: HTTP Messages, Methods, and Status Codes.....	1341
-----------------------------------------------------------	------

Chapter 82: HTTP Message Headers .....	1357
----------------------------------------	------

Chapter 83: HTTP Entities, Transfers, Coding Methods, and Content Management.....	1369
-----------------------------------------------------------------------------------	------

Chapter 84: HTTP Features, Capabilities, and Issues .....	1381
-----------------------------------------------------------	------

## **PART III-9: OTHER FILE AND MESSAGE TRANSFER APPLICATIONS**

Chapter 85: Usenet (Network News) and the TCP/IP Network News Transfer Protocol (NNTP) .....	1397
----------------------------------------------------------------------------------------------	------

Chapter 86: Gopher Protocol (Gopher) .....	1431
--------------------------------------------	------

## **PART III-10: INTERACTIVE AND ADMINISTRATIVE UTILITIES AND PROTOCOLS**

Chapter 87: TCP/IP Interactive and Remote Application Protocols .....	1437
Chapter 88: TCP/IP Administration and Troubleshooting Utilities and Protocols .....	1461
<b>INDEX.....</b>	<b>1491</b>
<b>RFCs BY NUMBER .....</b>	<b>1537</b>

# CONTENTS IN DETAIL

LIST OF FIGURES .....	xlv
LIST OF TABLES .....	lv
ACKNOWLEDGMENTS .....	lxv
ABOUT THE AUTHOR .....	lxvi
INTRODUCTION .....	lxix
Goals of <i>The TCP/IP Guide</i> .....	lxix
Scope of <i>The TCP/IP Guide</i> .....	lxxi
<i>The TCP/IP Guide</i> Features .....	lxxii
The <i>TCP/IP Guide</i> Online! .....	lxxiii
Your Feedback and Suggestions .....	lxxiii

## SECTION I TCP/IP OVERVIEW AND BACKGROUND INFORMATION

PART I-1 NETWORKING FUNDAMENTALS	3
----------------------------------	---

<b>1 Networking Introduction, Characteristics, and Types</b>	<b>5</b>
Introduction to Networking .....	6
What Is Networking? .....	6
The Advantages and Benefits of Networking .....	7
The Disadvantages and Costs of Networking .....	8
Fundamental Network Characteristics .....	10
Networking Layers, Models, and Architectures .....	10
Protocols: What Are They, Anyway? .....	11
Circuit-Switching and Packet-Switching Networks .....	13
Connection-Oriented and Connectionless Protocols .....	15
Messages: Packets, Frames, Datagrams, and Cells .....	17
Message Formatting: Headers, Payloads, and Footers .....	19
Message Addressing and Transmission Methods: Unicast, Broadcast, and Multicast .....	20
Network Structural Models and Client-Server and Peer-to-Peer Networking .....	23
Types and Sizes of Networks .....	25
Segments, Networks, Subnetworks, and Internetworks .....	27
The Internet, Intranets, and Extranets .....	29
<b>2 Network Performance Issues and Concepts</b>	<b>31</b>
Putting Network Performance in Perspective .....	32
Balancing Network Performance with Key Nonperformance Characteristics .....	33
Performance Measurements: Speed, Bandwidth, Throughput, and Latency .....	34
Speed .....	34
Bandwidth .....	35
Throughput .....	35
Latency .....	35
Summary of Performance Measurements .....	36
Understanding Performance Measurement Units .....	37
Bits and Bytes .....	37
Baud .....	38

Theoretical and Real-World Throughput, and Factors Affecting Network Performance .....	39
Normal Network Overhead .....	39
External Performance Limiters .....	40
Network Configuration Problems .....	40
Asymmetry .....	41
Simplex, Full-Duplex, and Half-Duplex Operation .....	41
Simplex Operation .....	42
Half-Duplex Operation .....	42
Full-Duplex Operation .....	42
Quality of Service (QoS) .....	43
<b>3 Network Standards and Standards Organizations</b>	<b>45</b>
Proprietary, Open, and De Facto Standards .....	46
Proprietary Standards .....	46
Open Standards .....	47
De Facto Standards .....	48
Networking Standards .....	48
International Networking Standards Organizations .....	49
Networking Industry Groups .....	51
Internet Standards Organizations (ISOC, IAB, IESG, IETF, IRSG, and IRTF) .....	52
Internet Registration Authorities and Registries (IANA, ICANN, APNIC, ARIN, LACNIC, and RIPE NCC) .....	55
Internet Centralized Registration Authorities .....	55
Modern Hierarchy of Registration Authorities .....	56
Internet Standards and the Request for Comment (RFC) Process .....	57
RFC Categories .....	58
The Internet Standardization Process .....	58
<b>4 A Review of Data Representation and the Mathematics of Computing</b>	<b>61</b>
Binary Information and Representation: Bits, Bytes, Nibbles, Octets, and Characters .....	62
Binary Information .....	62
Binary Information Representation and Groups .....	63
Byte Versus Octet .....	64
Decimal, Binary, Octal, and Hexadecimal Numbers .....	65
Binary Numbers and Their Decimal Equivalents .....	65
Making Binary Numbers Easier to Use by Grouping Bits .....	66
Octal Numbers .....	66
Hexadecimal Numbers .....	67
Decimal, Binary, Octal, and Hexadecimal Number Conversion .....	68
Binary, Octal, and Hexadecimal Conversions .....	68
Conversion from Binary, Octal, or Hexadecimal to Decimal .....	69
Conversion from Decimal to Binary, Octal, or Hexadecimal .....	70
Binary, Octal, and Hexadecimal Arithmetic .....	71
Binary Arithmetic .....	72
Octal and Hexadecimal Arithmetic .....	72
Boolean Logic and Logical Functions .....	73
Boolean Logical Functions .....	73
Combining Boolean Expressions .....	75
Bit Masking (Setting, Clearing, and Inverting) Using Boolean Logical Functions .....	75
Setting Groups of Bits with OR .....	76
Clearing Bits with AND .....	76
Inverting Bits with XOR .....	77

<b>5 General OSI Reference Model Issues and Concepts</b>	<b>81</b>
History of the OSI Reference Model .....	82
General Reference Model Issues .....	83
The Benefits of Networking Models .....	83
Why Understanding the OSI Reference Model Is Important to You .....	84
How to Use the OSI Reference Model .....	85
Other Network Architectures and Protocol Stacks .....	86
Key OSI Reference Model Concepts .....	87
OSI Reference Model Networking Layers, Sublayers, and Layer Groupings .....	87
“N” Notation and Other OSI Model Layer Terminology .....	89
Interfaces: Vertical (Adjacent Layer) Communication .....	91
Protocols: Horizontal (Corresponding Layer) Communication .....	93
Data Encapsulation, Protocol Data Units (PDUs), and Service Data Units (SDUs) .....	95
Indirect Device Connection and Message Routing .....	98
<b>6 OSI Reference Model Layers</b>	<b>101</b>
Physical Layer (Layer 1) .....	102
Data Link Layer (Layer 2) .....	103
Network Layer (Layer 3) .....	105
Transport Layer (Layer 4) .....	106
Session Layer (Layer 5) .....	109
Presentation Layer (Layer 6) .....	110
Application Layer (Layer 7) .....	111
<b>7 OSI Reference Model Summary</b>	<b>113</b>
Understanding the OSI Model: An Analogy .....	113
Remembering the OSI Model Layers: Some Mnemonics .....	116
Summarizing the OSI Model Layers: A Summary Chart .....	117
<b>PART I-3 TCP/IP PROTOCOL SUITE AND ARCHITECTURE</b>	<b>119</b>
<b>8 TCP/IP Protocol Suite and Architecture</b>	<b>121</b>
TCP/IP Overview and History .....	122
TCP/IP History and Development .....	122
Important Factors in the Success of TCP/IP .....	123
TCP/IP Services .....	125
The TCP/IP Client/Server Structural Model .....	125
Hardware and Software Roles .....	127
Transactional Roles .....	127
TCP/IP Architecture and the TCP/IP Model .....	128
Network Interface Layer .....	128
Internet Layer .....	129
Host-to-Host Transport Layer .....	130
Application Layer .....	130
TCP/IP Protocols .....	131

## **SECTION II**

### **TCP/IP LOWER-LAYER CORE PROTOCOLS**

#### **PART II-1 TCP/IP NETWORK INTERFACE LAYER PROTOCOLS 137**

<b>9 TCP/IP Serial Line Internet Protocol (SLIP) and Point-to-Point Protocol (PPP)</b>	<b>139</b>
Overview and Fundamentals	
SLIP versus PPP .....	140
Serial Line Internet Protocol (SLIP) .....	141
SLIP Data Framing Method and General Operation .....	141
Problems and Limitations of SLIP .....	142
Point-to-Point Protocol (PPP) Overview and Fundamentals .....	144
Development and Standardization .....	144
Function and Architecture .....	145
Advantages and Benefits .....	145
PPP Main Components .....	146
PPP Functional Groups .....	147
General Operation .....	147
PPP Link Setup and Phases .....	148
PPP Standards .....	151
<b>10 PPP Core Protocols: Link Control, Network Control, and Authentication</b>	<b>155</b>
Link Control Protocol (LCP) .....	155
LCP Packets .....	156
LCP Link Configuration .....	157
LCP Link Maintenance .....	159
LCP Link Termination .....	159
Other LCP Messages .....	159
The Network Control Protocols (IPCP, IPXCP, NBFCP, and Others) .....	159
Operation of NCPs .....	160
The Internet Protocol Control Protocol (IPCP): An Example NCP .....	162
PPP Authentication Protocols: PAP and CHAP .....	162
PAP .....	162
CHAP .....	163
<b>11 PPP Feature Protocols</b>	<b>167</b>
PPP Link Quality Monitoring and Reporting (LQM, LQR) .....	168
LQR Setup .....	168
Using Link Quality Reports .....	169
PPP Compression Control Protocol (CCP) and Compression Algorithms .....	169
CCP Operation: Compression Setup .....	170
CCP Configuration Options and Compression Algorithms .....	171
Compression Algorithm Operation: Compressing and Decompressing Data .....	171
PPP Encryption Control Protocol (ECP) and Encryption Algorithms .....	172
ECP Operation: Encryption Setup .....	173
ECP Configuration Options and Encryption Algorithms .....	173
Encryption Algorithm Operation: Encrypting and Decrypting Data .....	174
PPP Multilink Protocol (MP, MLP, MLPPP) .....	175
PPP Multilink Protocol Architecture .....	176
PPP Multilink Protocol Setup and Configuration .....	177
PPP Multilink Protocol Operation .....	177

PPP Bandwidth Allocation Protocol (BAP) and Bandwidth Allocation Control Protocol (BACP) .....	178
BACP Operation: Configuring the Use of BAP .....	179
BAP Operation: Adding and Removing Links .....	179

## **12 PPP Protocol Frame Formats 181**

PPP General Frame Format .....	182
Protocol Field Ranges .....	183
Protocol Field Values .....	184
PPP Field Compression .....	185
PPP General Control Protocol Frame Format and Option Format .....	186
PPP Control Messages and Code Values .....	187
PPP Control Message Option Format .....	188
Summary of PPP Control Message Formatting .....	190
PPP Link Control Protocol (LCP) Frame Formats .....	190
PAP and CHAP Frame Formats .....	192
PPP PAP Control Frame Formats .....	192
PPP CHAP Control Frame Formats .....	194
PPP Multilink Protocol (MP) Frame Format .....	195
PPP MP Frame Fragmentation Process .....	196
PPP MP Fragment Frame Format .....	196
PPP MP Fragmentation Demonstration .....	198

## **PART II-2 TCP/IP NETWORK INTERFACE/INTERNET LAYER CONNECTION PROTOCOLS 201**

### **13 Address Resolution and the TCP/IP Address Resolution Protocol (ARP) 203**

Address Resolution Concepts and Issues .....	204
The Need for Address Resolution .....	204
Address Resolution Through Direct Mapping .....	206
Dynamic Address Resolution .....	209
TCP/IP Address Resolution Protocol (ARP) .....	212
ARP Address Specification and General Operation .....	213
ARP Message Format .....	216
ARP Caching .....	218
Proxy ARP .....	221
TCP/IP Address Resolution for IP Multicast Addresses .....	223
TCP/IP Address Resolution for IP Version 6 .....	224

### **14 Reverse Address Resolution and the TCP/IP Reverse Address Resolution Protocol (RARP) 227**

The Reverse Address Resolution Protocol (RARP) .....	228
RARP General Operation .....	229
Limitations of RARP .....	231

## **PART II-3 INTERNET PROTOCOL VERSION 4 (IP/IPv4) 233**

### **15 Internet Protocol Versions, Concepts, and Overview 235**

IP Overview and Key Operational Characteristics .....	236
IP Functions .....	238

IP History, Standards, Versions, and Closely Related Protocols .....	239
IP Versions and Version Numbers .....	239
IP-Related Protocols .....	240
<b>16 IPv4 Addressing Concepts and Issues</b>	<b>241</b>
IP Addressing Overview and Fundamentals .....	242
Number of IP Addresses Per Device .....	243
Address Uniqueness and Network Specificity .....	243
Contrasting IP Addresses and Data Link Layer Addresses .....	244
Private and Public IP Network Addresses .....	244
IP Address Configuration and Addressing Types .....	244
IP Address Size, Address Space, and Notation .....	245
IP Address Size and Binary Notation .....	245
IP Address Dotted Decimal Notation .....	245
IP Address Space .....	246
IP Basic Address Structure and Main Components .....	247
Network ID and Host ID .....	247
Location of the Division Between Network ID and Host ID .....	248
IP Addressing Categories and IP Address Adjuncts .....	249
Conventional (Classful) Addressing .....	250
Subnetted Classful Addressing .....	250
Classless Addressing .....	250
Subnet Mask and Default Gateway .....	251
Number of IP Addresses and Multihoming .....	251
IP Address Management and Assignment Methods and Authorities .....	252
<b>17 Classful (Conventional) Addressing</b>	<b>255</b>
IP Classful Addressing Overview and Address Classes .....	256
IP Address Classes .....	256
Rationale for Classful Addressing .....	257
IP Classful Addressing Network and Host Identification and Address Ranges .....	258
Classful Addressing Class Determination Algorithm .....	258
Determining Address Class from the First Octet Bit Pattern .....	260
IP Address Class A, B, and C Network and Host Capacities .....	262
IP Addresses with Special Meanings .....	263
IP Reserved, Private, and Loopback Addresses .....	265
Reserved Addresses .....	265
Private, Unregistered, Nonroutable Addresses .....	265
Loopback Addresses .....	266
Reserved, Private, and Loopback Addressing Blocks .....	267
IP Multicast Addressing .....	268
Multicast Address Types and Ranges .....	268
Well-Known Multicast Addresses .....	269
Problems with Classful IP Addressing .....	269
<b>18 IP Subnet Addressing (Subnetting) Concepts</b>	<b>273</b>
IP Subnet Addressing Overview, Motivation, and Advantages .....	274
IP Subnetting: Three-Level Hierarchical IP Subnet Addressing .....	276
IP Subnet Masks, Notation, and Subnet Calculations .....	277
Function of the Subnet Mask .....	277
Subnet Mask Notation .....	278
Applying the Subnet Mask: An Example .....	279
Rationale for Subnet Mask Notation .....	280

IP Default Subnet Masks for Address Classes A, B, and C .....	281
IP Custom Subnet Masks .....	283
Deciding How Many Subnet Bits to Use .....	283
Determining the Custom Subnet Mask .....	284
Subtracting Two from the Number of Hosts per Subnet and (Possibly) Subnets per Network .....	285
IP Subnet Identifiers, Subnet Addresses, and Host Addresses .....	286
Subnet Identifiers .....	286
Subnet Addresses .....	287
Host Addresses Within Each Subnet .....	288
IP Subnetting Summary Tables for Class A, Class B, and Class C Networks .....	288
IP Variable Length Subnet Masking (VLSM) .....	292
The Solution: Variable Length Subnet Masking .....	294
Multiple-Level Subnetting Using VLSM .....	294
<b>19 IP Subnetting: Practical Subnet Design and Address Determination Example</b> ..... <b>297</b>	
IP Subnetting Step 1: Analyzing Requirements .....	298
IP Subnetting Step 2: Partitioning Network Address Host Bits .....	299
Class C Subnetting Design Example .....	300
Class B Subnetting Design Example .....	301
IP Subnetting Step 3: Determining the Custom Subnet Mask .....	302
Calculating the Custom Subnet Mask .....	303
Determining the Custom Subnet Mask Using Subnetting Tables .....	305
IP Subnetting Step 4: Determining Subnet Identifiers and Subnet Addresses .....	305
Class C Subnet ID and Address Determination Example .....	306
Class B Subnet ID and Address Determination Example .....	307
Using Subnet Address Formulas to Calculate Subnet Addresses .....	309
IP Subnetting Step 5: Determining Host Addresses for Each Subnet .....	310
Class C Host Address Determination Example .....	310
Class B Host Address Determination Example .....	313
Shortcuts for Computing Host Addresses .....	313
<b>20 IP Classless Addressing—Classless Inter-Domain Routing (CIDR)/Supernetting</b> ..... <b>315</b>	
IP Classless Addressing and Supernetting Overview .....	316
The Main Problem with Classful Addressing .....	316
The Solution: Eliminate Address Classes .....	317
The Many Benefits of Classless Addressing and Routing .....	317
IP Supernetting: CIDR Hierarchical Addressing and Notation .....	319
CIDR (Slash) Notation .....	319
Supernetting: Subnetting the Internet .....	320
Common Aspects of Classful and Classless Addressing .....	321
IP Classless Addressing Block Sizes and Classful Network Equivalents .....	322
IP CIDR Addressing Example .....	324
First Level of Division .....	324
Second Level of Division .....	327
Third Level of Division .....	327
<b>21 Internet Protocol Datagram Encapsulation and Formatting</b> ..... <b>329</b>	
IP Datagram Encapsulation .....	330
IP Datagram General Format .....	332
IP Datagram Time to Live (TTL) Field .....	335
IP Datagram Type of Service (TOS) Field .....	335
IP Datagram Options and Option Format .....	336

<b>22 IP Datagram Size, Fragmentation, and Reassembly</b>	<b>339</b>
IP Datagram Size, MTU, and Fragmentation Overview .....	340
IP Datagram Size and the Underlying Network Frame Size .....	340
MTU and Datagram Fragmentation .....	341
Multiple-Stage Fragmentation .....	342
Internet Minimum MTU: 576 Bytes .....	343
MTU Path Discovery .....	343
IP Message Fragmentation Process .....	344
The IP Fragmentation Process .....	344
Fragmentation-Related IP Datagram Header Fields .....	346
IP Message Reassembly .....	347
<b>23 IP Routing and Multicasting</b>	<b>351</b>
IP Datagram Delivery .....	352
Direct Datagram Delivery .....	353
Indirect Datagram Delivery (Routing) .....	353
The Relationship Between Datagram Routing and Addressing .....	354
IP Routing Concepts and the Process of Next-Hop Routing .....	355
IP Routes and Routing Tables .....	357
IP Routing in a Subnet or Classless Addressing (CIDR) Environment .....	359
IP Multicasting .....	360
Multicast Addressing .....	361
Multicast Group Management .....	361
Multicast Datagram Processing and Routing .....	361
<b>PART II-4 INTERNET PROTOCOL VERSION 6 (IPV6)</b>	<b>363</b>
<b>24 IPv6 Overview, Changes, and Transition</b>	<b>365</b>
IPv6 Motivation and Overview .....	366
IPv6 Standards .....	366
Design Goals of IPv6 .....	367
Major Changes and Additions in IPv6 .....	368
Transition from IPv4 to IPv6 .....	370
IPv4 to IPv6 Transition: Differences of Opinion .....	370
IPv4 to IPv6 Transition Methods .....	371
<b>25 IPv6 Addressing</b>	<b>373</b>
IPv6 Addressing Overview: Addressing Model, Address Types, and Address Size .....	374
IPv6 Addressing Model Characteristics .....	374
IPv6 Supported Address Types .....	375
IPv6 Address Size and Address Space .....	376
IPv6 Address and Address Notation and Prefix Representation .....	378
IPv6 Address Hexadecimal Notation .....	378
Zero Compression in IPv6 Addresses .....	379
IPv6 Mixed Notation .....	380
IPv6 Address Prefix Length Representation .....	381
IPv6 Address Space Allocation .....	381
IPv6 Global Unicast Address Format .....	383
Rationale for a Structured Unicast Address Block .....	383
Generic Division of the Unicast Address Space .....	384

IPv6 Implementation of the Unicast Address Space .....	384
Original Division of the Global Routing Prefix: Aggregators .....	385
A Sample Division of the Global Routing Prefix into Levels .....	386
IPv6 Interface Identifiers and Physical Address Mapping .....	388
IPv6 Special Addresses: Reserved, Private, Unspecified, and Loopback .....	389
Special Address Types .....	390
IPv6 Private Addresses Type Scopes .....	391
IPv6/IPv4 Address Embedding .....	392
IPv6 Multicast and Anycast Addressing .....	394
IPv6 Multicast Addresses .....	394
IPv6 Anycast Addresses .....	398
IPv6 Autoconfiguration and Renumbering .....	398
IPv6 Stateless Autoconfiguration .....	399
IPv6 Device Renumbering .....	400
<b>26 IPv6 Datagram Encapsulation and Formatting</b>	<b>401</b>
IPv6 Datagram Overview and General Structure .....	402
IPv6 Datagram Main Header Format .....	404
IPv6 Next Header Field .....	405
Key Changes to the Main Header Between IPv4 and IPv6 .....	406
IPv6 Datagram Extension Headers .....	407
IPv6 Header Chaining Using the Next Header Field .....	407
Summary of IPv6 Extension Headers .....	409
IPv6 Routing Extension Header .....	410
IPv6 Fragment Extension Header .....	411
IPv6 Extension Header Order .....	411
IPv6 Datagram Options .....	412
<b>27 IPv6 Datagram Size, Fragmentation, Reassembly, and Routing</b>	<b>415</b>
Overview of IPv6 Datagram Sizing and Fragmentation .....	416
Implications of IPv6's Source-Only Fragmentation Rule .....	417
The IPv6 Fragmentation Process .....	418
IPv6 Datagram Delivery and Routing .....	420
<b>PART II-5 IP-RELATED FEATURE PROTOCOLS</b>	<b>423</b>
<b>28 IP Network Address Translation (NAT) Protocol</b>	<b>425</b>
IP NAT Overview .....	426
Advantages of IP NAT .....	428
Disadvantages of IP NAT .....	429
IP NAT Address Terminology .....	430
IP NAT Static and Dynamic Address Mappings .....	433
Static Mappings .....	433
Dynamic Mappings .....	433
Choosing Between Static and Dynamic Mapping .....	433
IP NAT Unidirectional (Traditional/Outbound) Operation .....	434
IP NAT Bidirectional (Two-Way/Inbound) Operation .....	437
IP NAT Port-Based (Overloaded) Operation .....	439
IP NAT Overlapping/Twice NAT Operation .....	442
IP NAT Compatibility Issues and Special Handling Requirements .....	445

## **29 IP Security (IPsec) Protocols** 449

IPsec Overview, History, and Standards .....	450
Overview of IPsec Services and Functions .....	451
IPsec Standards .....	451
IPsec General Operation, Components, and Protocols .....	452
IPsec Core Protocols .....	453
IPsec Support Components .....	453
IPsec Architectures and Implementation Methods .....	454
Integrated Architecture .....	455
Bump in the Stack (BITS) Architecture .....	455
Bump in the Wire (BITW) Architecture .....	456
IPsec Modes: Transport and Tunnel .....	457
Transport Mode .....	457
Tunnel Mode .....	457
Comparing Transport and Tunnel Modes .....	457
IPsec Security Constructs .....	460
Security Policies, Security Associations, and Associated Databases .....	460
Selectors .....	461
Security Association Triples and Security Parameter Index (SPI) .....	461
IPsec Authentication Header (AH) .....	461
AH Datagram Placement and Linking .....	462
AH Format .....	465
IPsec Encapsulating Security Payload (ESP) .....	466
ESP Fields .....	466
ESP Operations and Field Use .....	467
ESP Format .....	470
IPsec Internet Key Exchange (IKE) .....	471
IKE Overview .....	472
IKE Operation .....	472

## **30 Internet Protocol Mobility Support (Mobile IP)** 475

Mobile IP Overview, History, and Motivation .....	476
The Problem with Mobile Nodes in TCP/IP .....	476
The Solution: Mobile IP .....	478
Limitations of Mobile IP .....	479
Mobile IP Concepts and General Operation .....	480
Mobile IP Device Roles .....	481
Mobile IP Functions .....	482
Mobile IP Addressing: Home and Care-Of Addresses .....	483
Foreign Agent Care-Of Address .....	484
Co-Located Care-Of Address .....	485
Advantages and Disadvantages of the Care-Of Address Types .....	485
Mobile IP Agent Discovery .....	486
Agent Discovery Process .....	486
Agent Advertisement and Agent Solicitation Messages .....	487
Mobile IP Home Agent Registration and Registration Messages .....	491
Mobile Node Registration Events .....	491
Registration Request and Registration Reply Messages .....	491
Registration Process .....	492
Registration Request Message Format .....	493
Registration Reply Message Format .....	495
Mobile IP Data Encapsulation and Tunneling .....	495
Mobile IP Conventional Tunneling .....	496
Mobile IP Reverse Tunneling .....	498
Mobile IP and TCP/IP Address Resolution Protocol (ARP) Operation .....	498

Mobile IP Efficiency Issues .....	500
Mobile IP Security Considerations .....	503

## PART II-6 IP SUPPORT PROTOCOLS 505

<b>31 ICMP Concepts and General Operation</b>	<b>507</b>
ICMP Overview, History, Versions, and Standards .....	508
ICMP General Operation .....	510
The ICMP Message-Passing Service .....	510
ICMP Error Reporting Limited to the Datagram Source .....	511
ICMP Message Classes, Types, and Codes .....	512
ICMP Message Classes .....	512
ICMP Message Types .....	512
ICMP Message Codes .....	513
ICMP Message Class and Type Summary .....	513
ICMP Message Creation and Processing Conventions and Rules .....	515
Limitations on ICMP Message Responses .....	516
ICMP Message Processing Conventions .....	517
ICMP Common Message Format and Data Encapsulation .....	518
ICMP Common Message Format .....	518
Original Datagram Inclusion in ICMP Error Messages .....	519
ICMP Data Encapsulation .....	520
<b>32 ICMPv4 Error Message Types and Formats</b>	<b>521</b>
ICMPv4 Destination Unreachable Messages .....	522
ICMPv4 Destination Unreachable Message Format .....	522
ICMPv4 Destination Unreachable Message Subtypes .....	523
Interpretation of Destination Unreachable Messages .....	524
ICMPv4 Source Quench Messages .....	525
ICMPv4 Source Quench Message Format .....	526
Problems with Source Quench Messages .....	526
ICMPv4 Time Exceeded Messages .....	527
ICMPv4 Time Exceeded Message Format .....	528
Applications of Time Exceeded Messages .....	529
ICMPv4 Redirect Messages .....	530
ICMPv4 Redirect Message Format .....	530
Redirect Message Interpretation Codes .....	532
Limitations of Redirect Messages .....	532
ICMPv4 Parameter Problem Messages .....	533
ICMPv4 Parameter Problem Message Format .....	533
Parameter Problem Message Interpretation Codes and the Pointer Field .....	534
<b>33 ICMPv4 Informational Message Types and Formats</b>	<b>535</b>
ICMPv4 Echo (Request) and Echo Reply Messages .....	536
ICMPv4 Echo and Echo Reply Message Format .....	536
Application of Echo and Echo Reply Messages .....	537
ICMPv4 Timestamp (Request) and Timestamp Reply Messages .....	537
ICMPv4 Timestamp and Timestamp Reply Message Format .....	538
Issues Using Timestamp and Timestamp Reply Messages .....	539
ICMPv4 Router Advertisement and Router Solicitation Messages .....	539
The Router Discovery Process .....	540
ICMPv4 Router Advertisement Message Format .....	540

ICMPv4 Router Solicitation Message Format .....	542
Addressing and Use of Router Advertisement and Router Solicitation Messages .....	542
ICMPv4 Address Mask Request and Reply Messages .....	543
ICMPv4 Address Mask Request and Address Mask Reply Message Format .....	543
Use of Address Mask Request and Address Mask Reply Messages .....	544
ICMPv4 Traceroute Messages .....	544
ICMPv4 Traceroute Message Format .....	545
Use of Traceroute Messages .....	546
<b>34 ICMPv6 Error Message Types and Formats</b>	<b>547</b>
ICMPv6 Destination Unreachable Messages .....	548
ICMPv6 Destination Unreachable Message Format .....	548
ICMPv6 Destination Unreachable Message Subtypes .....	549
Processing of Destination Unreachable Messages .....	550
ICMPv6 Packet Too Big Messages .....	550
ICMPv6 Packet Too Big Message Format .....	550
Applications of Packet Too Big Messages .....	551
ICMPv6 Time Exceeded Messages .....	552
ICMPv6 Time Exceeded Message Format .....	553
Applications of Time Exceeded Messages .....	554
ICMPv6 Parameter Problem Messages .....	554
ICMPv6 Parameter Problem Message Format .....	555
Parameter Problem Message Interpretation Codes and the Pointer Field .....	555
<b>35 ICMPv6 Informational Message Types and Formats</b>	<b>557</b>
ICMPv6 Echo Request and Echo Reply Messages .....	558
ICMPv6 Echo and Echo Reply Message Format .....	558
Application of Echo and Echo Reply Messages .....	559
ICMPv6 Router Advertisement and Router Solicitation Messages .....	560
ICMPv6 Router Advertisement Message Format .....	560
ICMPv6 Router Solicitation Message Format .....	562
Addressing of Router Advertisement and Router Solicitation Messages .....	562
ICMPv6 Neighbor Advertisement and Neighbor Solicitation Messages .....	563
ICMPv6 Neighbor Advertisement Message Format .....	563
ICMPv6 Neighbor Solicitation Message Format .....	564
Addressing of Neighbor Advertisement and Neighbor Solicitation Messages .....	565
ICMPv6 Redirect Messages .....	566
ICMPv6 Redirect Message Format .....	566
Application of Redirect Messages .....	568
ICMPv6 Router Renumbering Messages .....	568
IPv6 Router Renumbering .....	568
ICMPv6 Router Renumbering Message Format .....	569
Addressing of Router Renumbering Messages .....	571
ICMPv6 Informational Message Options .....	571
Source Link-Layer Address Option Format .....	571
Target Link-Layer Address Option Format .....	572
Prefix Information Option Format .....	572
Redirected Header Option Format .....	573
MTU Option Format .....	574
<b>36 IPv6 Neighbor Discovery (ND) Protocol</b>	<b>575</b>
IPv6 ND Overview .....	576
Formalizing Local Network Functions: The Neighbor Concept .....	577
Neighbor Discovery Standards .....	577

IPv6 ND General Operational Overview .....	578
Host-Router Discovery Functions .....	579
Host-Host Communication Functions .....	579
Redirect Function .....	579
Relationships Between Functions .....	580
ICMPv6 Messages Used by ND .....	580
IPv6 ND Functions Compared to Equivalent IPv4 Functions .....	580
IPv6 ND Host-Router Discovery Functions .....	582
Host-Router Discovery Functions Performed by Routers .....	582
Host-Router Discovery Functions Performed by Hosts .....	583
IPv6 ND Host-Host Communication Functions .....	583
Next-Hop Determination .....	584
Address Resolution .....	584
Updating Neighbors Using Neighbor Advertisement Messages .....	585
Neighbor Unreachability Detection and the Neighbor Cache .....	585
Duplicate Address Detection .....	586
IPv6 ND Redirect Function .....	586

## **PART II-7 TCP/IP ROUTING PROTOCOLS (GATEWAY PROTOCOLS)                   589**

### **37 Overview of Key Routing Protocol Concepts                   591**

Routing Protocol Architectures .....	591
Core Architecture .....	592
Autonomous System (AS) Architecture .....	592
Modern Protocol Types: Interior and Exterior Routing Protocols .....	593
Routing Protocol Algorithms and Metrics .....	594
Distance-Vector (Bellman-Ford) Routing Protocol Algorithm .....	594
Link-State (Shortest-Path First) Routing Protocol Algorithm .....	595
Hybrid Routing Protocol Algorithms .....	595
Static and Dynamic Routing Protocols .....	595

### **38 Routing Information Protocol (RIP, RIP-2, and RIPng)                   597**

RIP Overview .....	598
RIP Standardization .....	598
RIP Operational Overview, Advantages, and Limitations .....	599
Development of RIP Version 2 (RIP-2) and RIPng for IPv6 .....	600
RIP Route Determination Algorithm and Metric .....	600
RIP Routing Information and Route Distance Metric .....	600
RIP Route Determination Algorithm .....	601
RIP Route Determination and Information Propagation .....	601
Default Routes .....	604
RIP General Operation, Messaging, and Timers .....	604
RIP Messages and Basic Message Types .....	604
RIP Update Messaging and the 30-Second Timer .....	605
Preventing Stale Information: The Timeout Timer .....	605
Removing Stale Information: The Garbage-Collection Timer .....	606
Triggered Updates .....	606
RIP Problems and Some Resolutions .....	606
Issues with RIP's Algorithm .....	607
Issues with RIP's Metric .....	610
RIP Special Features for Resolving RIP Algorithm Problems .....	610
RIP Version-Specific Message Formats and Features .....	614
RIP Version 1 (RIP-1) Message Format and Features .....	614
RIP Version 2 (RIP-2) Message Format and Features .....	617
RIPng (RIPv6) Message Format and Features .....	620

## **39 Open Shortest Path First (OSPF)** 625

OSPF Overview .....	626
Development and Standardization of OSPF .....	626
Overview of OSPF Operation .....	627
OSPF Features and Drawbacks .....	627
OSPF Basic Topology and the Link-State Database (LSDB) .....	628
OSPF Basic Topology .....	628
LSDB Information Storage and Propagation .....	629
OSPF Hierarchical Topology .....	630
OSPF Areas .....	630
Router Roles in OSPF Hierarchical Topology .....	631
OSPF Route Determination Using SPF Trees .....	633
The SPF Tree .....	633
OSPF Route Determination .....	634
OSPF General Operation .....	637
OSPF Message Types .....	638
OSPF Messaging .....	638
OSPF Message Authentication .....	639
OSPF Message Formats .....	639
OSPF Common Header Format .....	639
OSPF Hello Message Format .....	641
OSPF Database Description Message Format .....	641
OSPF Link State Request Message Format .....	643
OSPF Link State Update Message Format .....	643
OSPF Link State Acknowledgment Message Format .....	644
OSPF Link State Advertisements and the LSA Header Format .....	644

## **40 Border Gateway Protocol (BGP/BGP-4)** 647

BGP Overview .....	648
BGP Versions and Defining Standards .....	649
Overview of BGP Functions and Features .....	650
BGP Topology .....	651
BGP Speakers, Router Roles, Neighbors, and Peers .....	652
BGP AS Types, Traffic Flows, and Routing Policies .....	653
BGP Route Storage and Advertisement .....	656
BGP Route Information Management Functions .....	656
BGP Routing Information Bases (RIBs) .....	656
BGP Path Attributes and Algorithm Overview .....	657
BGP Path Attribute Classes .....	658
BGP Path Attribute Characteristics .....	659
BGP Route Determination and the BGP Decision Process .....	659
BGP Decision Process Phases .....	660
Criteria for Assigning Preferences to Routes .....	660
Limitations on BGP's Ability to Select Efficient Routes .....	661
Originating New Routes and Withdrawing Unreachable Routes .....	661
BGP General Operation and Messaging .....	662
Speaker Designation and Connection Establishment .....	662
Route Information Exchange .....	662
Connectivity Maintenance .....	663
Error Reporting .....	663
BGP Detailed Messaging, Operation, and Message Formats .....	663
BGP Message Generation and Transport .....	663
BGP General Message Format .....	664
BGP Connection Establishment: Open Messages .....	666
BGP Route Information Exchange: Update Messages .....	667
BGP Connectivity Maintenance: Keepalive Messages .....	672
BGP Error Reporting: Notification Messages .....	673

<b>41 Other Routing Protocols</b>	<b>677</b>
TCP/IP Gateway-to-Gateway Protocol (GGP) .....	678
The HELLO Protocol (HELLO) .....	679
Interior Gateway Routing Protocol (IGRP) .....	681
Enhanced Interior Gateway Routing Protocol (EIGRP) .....	682
TCP/IP Exterior Gateway Protocol (EGP) .....	684
<b>PART II-8 TCP/IP TRANSPORT LAYER PROTOCOLS</b>	<b>687</b>
<b>42 Overview and Comparison of TCP and UDP</b>	<b>689</b>
Two Protocols for TCP/IP Transport Layer Requirements .....	690
Applications of TCP and UDP .....	691
TCP Applications .....	691
UDP Applications .....	692
Summary Comparison of UDP and TCP .....	692
<b>43 TCP and UDP Addressing: Ports and Sockets</b>	<b>695</b>
TCP/IP Processes, Multiplexing, and Client/Server Application Roles .....	696
Multiplexing and Demultiplexing .....	696
TCP/IP Client Processes and Server Processes .....	697
TCP/IP Ports: TCP/UDP Addressing .....	698
Multiplexing and Demultiplexing Using Ports .....	699
Source Port and Destination Port Numbers .....	699
Summary of Port Use for Datagram Transmission and Reception .....	701
TCP/IP Application Assignments and Server Port Number Ranges .....	701
Reserved Port Numbers .....	702
TCP/UDP Port Number Ranges .....	702
TCP/IP Client (Ephemeral) Ports and Client/Server Application Port Use .....	703
Ephemeral Port Number Assignment .....	704
Ephemeral Port Number Ranges .....	704
Port Number Use During a Client/Server Exchange .....	705
TCP/IP Sockets and Socket Pairs: Process and Connection Identification .....	706
Common TCP/IP Applications and Well-Known and Registered Port Numbers .....	707
<b>44 TCP/IP User Datagram Protocol (UDP)</b>	<b>711</b>
UDP Overview, History, and Standards .....	712
UDP Operation .....	713
What UDP Does .....	713
What UDP Does Not Do .....	713
UDP Message Format .....	714
UDP Common Applications and Server Port Assignments .....	716
Why Some TCP/IP Applications Use UDP .....	716
Common UDP Applications and Server Port Use .....	717
Applications That Use Both UDP and TCP .....	718
<b>45 TCP Overview, Functions, and Characteristics</b>	<b>719</b>
TCP Overview, History, and Standards .....	720
TCP History .....	720
Overview of TCP Operation .....	721
TCP Standards .....	721

TCP Functions .....	722
Functions That TCP Performs .....	723
Functions That TCP Doesn't Perform .....	723
TCP Characteristics .....	724
The Robustness Principle .....	726
<b>46 Transmission Control Protocol (TCP) Fundamentals and General Operation</b>	<b>727</b>
TCP Data Handling and Processing .....	728
Increasing the Flexibility of Application Data Handling: TCP's Stream Orientation .....	728
TCP Data Packaging: Segments .....	728
TCP Data Identification: Sequence Numbers .....	729
The Need for Application Data Delimiting .....	731
TCP Sliding Window Acknowledgment System .....	731
The Problem with Unreliable Protocols: Lack of Feedback .....	732
Providing Basic Reliability Using Positive Acknowledgment with Retransmission (PAR) .....	732
Improving PAR .....	734
TCP's Stream-Oriented Sliding Window Acknowledgment System .....	734
More Information on TCP Sliding Windows .....	740
TCP Ports, Connections, and Connection Identification .....	741
TCP Common Applications and Server Port Assignments .....	742
<b>47 TCP Basic Operation: Connection Establishment, Management, and Termination</b>	<b>745</b>
TCP Operational Overview and the TCP Finite State Machine (FSM) .....	746
Basic FSM Concepts .....	746
The Simplified TCP FSM .....	747
TCP Connection Preparation .....	750
Storing Connection Data: The Transmission Control Block (TCB) .....	751
Active and Passive Opens .....	751
Preparation for Connection .....	752
TCP Connection Establishment Process: The Three-Way Handshake .....	752
Connection Establishment Functions .....	752
Control Messages Used for Connection Establishment: SYN and ACK .....	753
Normal Connection Establishment: The Three-Way Handshake .....	753
Simultaneous Open Connection Establishment .....	755
TCP Connection Establishment Sequence Number Synchronization and Parameter Exchange .....	757
Initial Sequence Number Selection .....	757
TCP Sequence Number Synchronization .....	758
TCP Parameter Exchange .....	759
TCP Connection Management and Problem Handling .....	760
The TCP Reset Function .....	760
Handling Reset Segments .....	761
Idle Connection Management and Keepalive Messages .....	761
TCP Connection Termination .....	762
Requirements and Issues In Connection Termination .....	762
Normal Connection Termination .....	763
The TIME-WAIT State .....	765
Simultaneous Connection Termination .....	766
<b>48 TCP Message Formatting and Data Transfer</b>	<b>769</b>
TCP Message (Segment) Format .....	770
TCP Checksum Calculation and the TCP Pseudo Header .....	774
Detecting Transmission Errors Using Checksums .....	774
Increasing the Scope of Detected Errors: The TCP Pseudo Header .....	774
Advantages of the Pseudo Header Method .....	776

TCP Maximum Segment Size (MSS) .....	777
MSS Selection .....	778
TCP Default MSS .....	778
Nondefault MSS Value Specification .....	779
TCP Sliding Window Data Transfer and Acknowledgment Mechanics .....	780
Sliding Window Transmit and Receive Categories .....	780
Send (SND) and Receive (RCV) Pointers .....	781
TCP Segment Fields Used to Exchange Pointer Information .....	783
An Example of TCP Sliding Window Mechanics .....	784
Real-World Complications of the Sliding Window Mechanism .....	789
TCP Immediate Data Transfer: Push Function .....	790
TCP Priority Data Transfer: Urgent Function .....	791

49 TCP Reliability and Flow Control Features 793

TCP Segment Retransmission Timers and the Retransmission Queue .....	794
Managing Retransmissions Using the Retransmission Queue .....	794
Recognizing When a Segment Is Fully Acknowledged .....	795
TCP Noncontiguous Acknowledgment Handling and Selective Acknowledgment (SACK) .....	798
Policies for Dealing with Outstanding Unacknowledged Segments .....	799
A Better Solution: Selective Acknowledgment (SACK) .....	801
TCP Adaptive Retransmission and Retransmission Timer Calculations .....	803
Adaptive Retransmission Based on RTT Calculations .....	803
Acknowledgment Ambiguity .....	804
Refinements to RTT Calculation and Karn's Algorithm .....	804
TCP Window Size Adjustment and Flow Control .....	805
Reducing Send Window Size to Reduce the Rate Data Is Sent .....	806
Reducing Send Window Size to Stop the Sending of New Data .....	808
Closing the Send Window .....	808
TCP Window Management Issues .....	809
Problems Associated with Shrinking the TCP Window .....	809
Reducing Buffer Size Without Shrinking the Window .....	810
Handling a Closed Window and Sending Probe Segments .....	811
TCP Silly Window Syndrome .....	812
How Silly Window Syndrome Occurs .....	812
Silly Window Syndrome Avoidance Algorithms .....	815
TCP Congestion Handling and Congestion Avoidance Algorithms .....	816
Congestion Considerations .....	816
TCP Congestion-Handling Mechanisms .....	817

## **SECTION III**

### **TCP/IP APPLICATION LAYER PROTOCOLS**

## PART III-1 NAME SYSTEMS AND TCP/IP NAME REGISTRATION AND NAME RESOLUTION

<b>50 Name System Issues, Concepts, and Techniques</b>	<b>825</b>
Name System Overview .....	826
Symbolic Names for Addressing .....	826
A Paradox: Name Systems Are Both Essential and Unnecessary .....	826
Factors That Determine the Necessity of a Name System .....	828
Basic Name System Functions: Name Space, Name Registration, and Name Resolution .....	829

Name Spaces and Name Architectures .....	831
Name Space Functions .....	831
Flat Name Architecture (Flat Name Space) .....	832
Hierarchical Name Architecture (Structured Name Space) .....	832
Comparing Name Architectures .....	833
Name Registration Methods, Administration, and Authorities .....	834
Name Registration Functions .....	834
Hierarchical Name Registration .....	835
Name Registration Methods .....	835
Name Resolution Techniques and Elements .....	836
Name Resolution Methods .....	837
Client/Server Name Resolution Functional Elements .....	838
Efficiency, Reliability, and Other Name Resolution Considerations .....	838
Efficiency Considerations .....	839
Reliability Considerations .....	839
Other Considerations .....	840
<b>51 TCP/IP Name Systems Overview and the Host Table Name System</b>	<b>841</b>
A Brief History of TCP/IP Host Names and Name Systems .....	842
Developing the First Name System: ARPAnet Host Name Lists .....	842
Storing Host Names in a Host Table File .....	842
Outgrowing the Host Table Name System and Moving to DNS .....	843
The TCP/IP Host Table Name System .....	843
Host Table Name Resolution .....	844
Host Table Name Registration .....	844
Weaknesses of the Host Table Name System .....	845
Use of the Host Table Name System in Modern Networking .....	846
<b>52 Domain Name System (DNS) Overview, Functions, and Characteristics</b>	<b>847</b>
DNS Overview, History, and Standards .....	848
Early DNS Development and the Move to Hierarchical Domains .....	848
Standardization of DNS and Initial Defining Standards .....	849
DNS Evolution and Important Additional Standards .....	850
DNS Adaptation for Internet Protocol Version 6 .....	850
DNS Design Goals, Objectives, and Assumptions .....	851
DNS Design Goals and Objectives .....	851
DNS Design Assumptions .....	852
DNS Components and General Functions .....	853
DNS Name Space .....	854
Name Registration (Including Administration and Authorities) .....	854
Name Resolution .....	854
<b>53 DNS Name Space, Architecture, and Terminology</b>	<b>857</b>
DNS Domains and the DNS Hierarchical Name Architecture .....	858
The Essential Concept in the DNS Name Space: Domains .....	858
The DNS Hierarchical Tree Structure of Names .....	858
DNS Structural Elements and Terminology .....	860
DNS Tree-Related Terminology .....	860
DNS Domain-Related Terminology .....	860
DNS Family-Related Terminology .....	861
DNS Labels, Names, and Syntax Rules .....	863
DNS Labels and Label Syntax Rules .....	863
Domain Name Construction .....	864

Absolute (Fully Qualified) and Relative (Partially Qualified) Domain Name Specifications .....	865
Fully Qualified Domain Names .....	865
Partially Qualified Domain Names .....	866
<b>54 DNS Name Registration, Public Administration, Zones, and Authorities</b>	<b>867</b>
DNS Hierarchical Authority Structure and the Distributed Name Database .....	868
The DNS Root Domain Central Authority .....	868
TLD Authorities .....	869
Lower-Level Authority Delegation .....	869
Authority Hierarchy's Relationship to the Name Hierarchy .....	869
The DNS Distributed Name Database .....	869
DNS Organizational (Generic) TLDs and Authorities .....	870
Original Generic TLDs .....	870
New Generic TLDs .....	871
DNS Geopolitical (Country Code) TLDs and Authorities .....	874
Country Code Designations .....	874
Country Code TLD Authorities .....	875
Leasing/Sale of Country Code Domains .....	875
Drawbacks of the Geopolitical TLDs .....	876
Public Registration for Second-Level and Lower Domains .....	876
Registration Authority .....	877
Registration Coordination .....	878
DNS Public Registration Disputes and Dispute Resolution .....	878
Public Registration Disputes .....	878
Methods of Registration Dispute Resolution .....	880
The Uniform Domain Name Dispute Resolution Policy .....	880
DNS Name Space Administrative Hierarchy Partitioning: DNS Zones of Authority .....	881
Methods of Dividing a Name Space into Zones of Authority .....	882
The Impact of Zones on Name Resolution: Authoritative Servers .....	882
DNS Private Name Registration .....	884
Using Publicly Accessible Private Names .....	884
Using Private Names for Internal Use .....	885
Using Private Names on Networks Not Connected to the Internet .....	885
<b>55 DNS Name Server Concepts and Operation</b>	<b>887</b>
DNS General Operation .....	888
DNS Name Server Architecture and the Distributed Name Database .....	888
DNS Server Support Functions .....	889
The Logical Nature of the DNS Name Server Hierarchy .....	890
DNS Name Server Data Storage .....	890
Binary and Text Representations of Resource Records .....	890
Use of RRs and Master Files .....	891
Common RR Types .....	892
RR Classes .....	893
DNS Name Server Types and Roles .....	893
Master (Primary)/Slave (Secondary) Servers .....	893
Name Server Roles .....	895
Caching-Only Name Servers .....	895
DNS Zone Management, Contacts, and Zone Transfers .....	895
Domain Contacts .....	896
Zone Transfers .....	896
DNS Root Name Servers .....	899
Root Name Server Redundancy .....	899
Current Root Name Servers .....	900

DNS Name Server Caching .....	901
Name Server Caching .....	902
Caching Data Persistence and the Time to Live Interval .....	902
Negative Caching .....	904
DNS Name Server Load Balancing .....	904
Using Multiple Address Records to Spread Out Requests to a Domain .....	904
Using Multiple DNS Servers to Spread Out DNS Requests .....	905
DNS Name Server Enhancements .....	905
Automating Zone Transfers: DNS Notify .....	906
Improving Zone Transfer Efficiency: Incremental Transfers .....	907
Dealing with Dynamic IP Addresses: DNS Update/Dynamic DNS .....	907
<b>56 DNS Resolution Concepts and Resolver Operations</b>	<b>909</b>
DNS Resolver Functions and General Operation .....	910
Name Resolution Services .....	910
Functions Performed by Name Resolvers .....	910
DNS Name Resolution Techniques: Iterative and Recursive Resolution .....	911
Iterative Resolution .....	912
Recursive Resolution .....	913
Contrasting Iterative and Recursive Resolution .....	913
DNS Name Resolution Efficiency Improvements: Caching and Local Resolution .....	915
The Motivation for Caching: Locality of Reference .....	915
Name Resolver Caching .....	916
Local Resolution .....	916
DNS Name Resolution Process .....	917
A Simple Example of DNS Name Resolution .....	917
Changes to Resolution to Handle Aliases (CNAME Records) .....	920
DNS Reverse Name Resolution Using the IN-ADDR.ARPA Domain .....	920
The Original Method: Inverse Querying .....	921
The IN-ADDR.ARPA Name Structure for Reverse Resolution .....	921
RR Setup for Reverse Resolution .....	922
DNS Electronic Mail Support and Mail Exchange (MX) Resource Records .....	924
Special Requirements for Email Name Resolution .....	924
The Mail Exchange (MX) Record and Its Use .....	925
<b>57 DNS Messaging and Message, Resource Record, and Master File Formats</b>	<b>927</b>
DNS Message Generation and Transport .....	928
DNS Client/Server Messaging Overview .....	928
DNS Message Transport Using UDP and TCP .....	929
DNS Message Processing and General Message Format .....	930
DNS Message Header Format .....	932
DNS Question Section Format .....	935
DNS Message Resource Record Field Formats .....	935
DNS Common RR Format .....	936
RData Field Formats for Common RRs .....	936
DNS Name Notation and Message Compression .....	940
Standard DNS Name Notation .....	940
DNS Electronic Mail Address Notation .....	941
DNS Message Compression .....	941
DNS Master File Format .....	943
DNS Common Master File Record Format .....	944
Use and Interpretation of Partially Qualified Domain Names (PQDNs) .....	944
Master File Directives .....	945
Syntax Rules for Master Files .....	945
Specific RR Syntax and Examples .....	946
Sample Master File .....	948

DNS Changes to Support IPv6 .....	948
IPv6 DNS Extensions .....	949
Proposed Changes to the IPv6 DNS Extensions .....	949

## **PART III-2 NETWORK FILE AND RESOURCE SHARING PROTOCOLS 951**

<b>58 Network File and Resource Sharing and the TCP/IP Network File System (NFS) 953</b>	
File and Resource Sharing Concepts and Components .....	954
The Power of File and Resource Sharing Protocols .....	954
Components of a File and Resource Sharing Protocol .....	954
NFS Design Goals, Versions, and Standards .....	955
NFS Design Goals .....	955
NFS Versions and Standards .....	956
NFS Architecture and Components .....	957
NFS Main Components .....	957
Other Important NFS Functions .....	958
NFS Data Definition with the External Data Representation (XDR) Standard .....	959
A Method of Universal Data Exchange: XDR .....	959
XDR Data Types .....	960
NFS Client/Server Operation Using Remote Procedure Calls (RPCs) .....	961
RPC Operation and Transport Protocol Usage .....	962
Client and Server Responsibilities in NFS .....	963
Client and Server Caching .....	963
NFS Server Procedures and Operations .....	964
NFS Version 2 and Version 3 Server Procedures .....	964
NFS Version 4 Server Procedures and Operations .....	966
NFS File System Model and the Mount Protocol .....	968
The NFS File System Model .....	968
The Mount Protocol .....	968

## **PART III-3 HOST CONFIGURATION AND TCP/IP HOST CONFIGURATION PROTOCOLS 971**

<b>59 Host Configuration Concepts, Issues, and Motivation 973</b>	
The Purpose of Host Configuration .....	973
The Problems with Manual Host Configuration .....	974
Automating the Process: Host Configuration Protocols .....	975
The Role of Host Configuration Protocols in TCP/IP .....	975

<b>60 TCP/IP Bootstrap Protocol (BOOTP) 977</b>	
BOOTP Overview, History, and Standards .....	978
BOOTP: Correcting the Weaknesses of RARP .....	978
Vendor-Specific Parameters .....	979
Changes to BOOTP and the Development of DHCP .....	980
BOOTP Client/Server Messaging and Addressing .....	980
BOOTP Messaging and Transport .....	981
BOOTP Use of Broadcasts and Ports .....	981
Retransmission of Lost Messages .....	982
BOOTP Detailed Operation .....	983
BOOTP Bootstrapping Procedure .....	983
Interpretation of the Client IP Address (CIAddr) Field .....	984
BOOTP Message Format .....	985

BOOTP Vendor-Specific Area and Vendor Information Extensions .....	988
BOOTP Vendor Information Extensions .....	989
BOOTP Vendor Information Fields .....	990
BOOTP Relay Agents (Forwarding Agents) .....	991
The Function of BOOTP Relay Agents .....	992
Normal BOOTP Operation Using a Relay Agent .....	993
Relaying BOOTP Requests Using Broadcasts .....	994
<b>61 DHCP Overview and Address Allocation Concepts</b>	<b>997</b>
DHCP Overview, History, and Standards .....	998
DHCP: Building on BOOTP's Strengths .....	999
Overview of DHCP Features .....	999
DHCP Address Assignment and Allocation Mechanisms .....	1000
DHCP Address Allocation .....	1000
DHCP Manual Allocation .....	1001
DHCP Dynamic Allocation .....	1001
DHCP Automatic Allocation .....	1002
DHCP Leases .....	1003
DHCP Lease Length Policy .....	1003
Issues with Infinite Leases .....	1005
DHCP Lease Life Cycle and Lease Timers .....	1005
DHCP Lease Life Cycle Phases .....	1006
Renewal and Rebinding Timers .....	1008
DHCP Lease Address Pools, Ranges, and Address Management .....	1008
Address Pool Size Selection .....	1009
Lease Address Ranges (Scopes) .....	1009
Other Issues with Address Management .....	1011
<b>62 DHCP Configuration and Operation</b>	<b>1013</b>
DHCP Overview of Client and Server Responsibilities .....	1014
DHCP Server Responsibilities .....	1014
DHCP Client Responsibilities .....	1015
DHCP Client/Server Roles .....	1015
DHCP Relay Agents .....	1016
DHCP Configuration Parameters, Storage, and Communication .....	1016
Configuration Parameter Management .....	1016
Parameter Storage .....	1017
Configuration Parameter Communication .....	1017
DHCP General Operation and the Client Finite State Machine .....	1017
DHCP Lease Allocation, Reallocation, and Renewal .....	1021
Initial Lease Allocation Process .....	1021
DHCP Lease Reallocation Process .....	1026
DHCP Lease Renewal and Rebinding Processes .....	1028
DHCP Early Lease Termination (Release) Process .....	1031
DHCP Parameter Configuration Process for Clients with Non-DHCP Addresses .....	1033
<b>63 DHCP Messaging, Message Types, and Formats</b>	<b>1035</b>
DHCP Message Generation, Addressing, Transport, and Retransmission .....	1036
Message Generation and General Formatting .....	1036
Message Transport .....	1036
Retransmission of Lost Messages .....	1037
DHCP Message Format .....	1038
DHCP Options .....	1041
Options and Option Format .....	1042

Option Categories .....	1043
Option Overloading .....	1044
Summary of DHCP Options/BOOTP Vendor Information Fields .....	1045
RFC 1497 Vendor Extensions .....	1045
IP Layer Parameters per Host .....	1046
IP Layer Parameters per Interface .....	1047
Link Layer Parameters per Interface .....	1048
TCP Parameters .....	1048
Application and Service Parameters .....	1048
DHCP Extensions .....	1050

## **64 DHCP Client/Server Implementation, Features, and IPv6 Support 1053**

DHCP Server and Client Implementation and Management Issues .....	1054
DHCP Server Implementations .....	1054
DHCP Client Implementations .....	1056
DHCP Message Relaying and BOOTP Relay Agents .....	1056
BOOTP Relay Agents for DHCP .....	1057
DHCP Relaying Process .....	1057
DHCP Autoconfiguration/Automatic Private IP Addressing (APIPA) .....	1058
APIPA Operation .....	1059
APIPA Limitations .....	1060
DHCP Server Conflict Detection .....	1061
DHCP and BOOTP Interoperability .....	1062
BOOTP Clients Connecting to a DHCP Server .....	1063
DHCP Clients Connecting to a BOOTP Server .....	1063
DHCP Security Issues .....	1063
DHCP Security Concerns .....	1064
DHCP Authentication .....	1064
DHCP for IP Version 6 (DHCPv6) .....	1065
Two Methods for Autoconfiguration in IPv6 .....	1065
DHCPv6 Operation Overview .....	1065
DHCPv6 Message Exchanges .....	1066

## **PART III-4 TCP/IP NETWORK MANAGEMENT FRAMEWORK AND PROTOCOLS 1067**

### **65 TCP/IP Internet Standard Management Framework Overview 1069**

Overview and History of the TCP/IP Internet Standard Management Framework and Simple Network Management Protocol (SNMP) .....	1070
Early Development of SNMP .....	1070
The Two Meanings of SNMP .....	1071
Design Goals of SNMP .....	1071
Further Development of SNMP and the Problem of SNMP Variations .....	1072
TCP/IP SNMP Operational Model, Components, and Terminology .....	1072
SNMP Device Types .....	1072
SNMP Entities .....	1073
SNMP Operational Model Summary .....	1073
TCP/IP Internet Standard Management Framework Architecture and Protocol Components .....	1075
SNMP Framework Components .....	1075
SNMP Framework Architecture .....	1076
TCP/IP Internet Standard Management Framework and SNMP Versions (SNMPv1, SNMPv2 Variants, and SNMPv3) .....	1076
SNMPv1 .....	1077
SNMPsec .....	1077

SNMPv2 .....	1078
SNMPv2 Variants .....	1078
SNMPv3 .....	1079
TCP/IP Internet Standard Management Framework and SNMP Standards .....	1079
<b>66 TCP/IP Structure of Management Information (SMI) and Management Information Bases (MIBs)</b>	<b>1083</b>
TCP/IP SMI and MIBs Overview .....	1084
SNMP's Information-Oriented Design .....	1084
MIB and MIB Objects .....	1085
Defining MIB Objects: SMI .....	1086
TCP/IP MIB Objects, Object Characteristics, and Object Types .....	1087
MIB Object Characteristics .....	1087
SMI Data Types .....	1089
TCP/IP MIB Object Descriptors and Identifiers and the Object Name Hierarchy .....	1090
Object Descriptors .....	1091
Object Identifiers .....	1091
Structure of the MIB Object Name Hierarchy .....	1092
Recursive Definition of MIB Object Identifiers .....	1094
TCP/IP MIB Modules and Object Groups .....	1094
The Organization of MIB Objects into Object Groups .....	1094
MIB Modules .....	1096
MIB Module Format .....	1097
<b>67 TCP/IP Simple Network Management Protocol (SNMP) Concepts and Operation</b>	<b>1099</b>
SNMP Protocol Overview .....	1100
Early Development of SNMPv1 .....	1100
SNMPv2 and the Division of SNMP into Protocol Operations and Transport Mappings .....	1101
SNMP Communication Methods .....	1102
SNMP Protocol Operations .....	1102
SNMP PDU Classes .....	1103
Basic Request/Response Information Poll Using GetRequest and (Get)Response Messages .....	1104
Table Traversal Using GetNextRequest and GetBulkRequest Messages .....	1105
Object Modification Using SetRequest Messages .....	1107
Information Notification Using Trap and InformRequest Messages .....	1109
SNMP Protocol Security Issues and Methods .....	1110
Problems with SNMPv1 Security .....	1111
SNMPv2/v3 Security Methods .....	1111
<b>68 SNMP Protocol Messaging and Message Formats</b>	<b>1113</b>
SNMP Protocol Message Generation .....	1114
SNMP Transport Mappings .....	1114
UDP Message Size Issues .....	1115
Lost Transmission Issues .....	1115
SNMP General Message Format .....	1116
The Difference Between SNMP Messages and PDUs .....	1117
General PDU Format .....	1117
SNMP Version 1 (SNMPv1) Message Format .....	1119
SNMPv1 General Message Format .....	1119
SNMPv1 PDU Formats .....	1120
SNMP Version 2 (SNMPv2) Message Formats .....	1122
SNMP Version 2 (SNMPv2p) Message Format .....	1123
Community-Based SNMP Version 2 (SNMPv2c) Message Format .....	1124

User-Based SNMP Version 2 (SNMPv2u) Message Format .....	1124
SNMPv2 PDU Formats .....	1126
SNMP Version 3 (SNMPv3) Message Format .....	1129
<b>69 TCP/IP Remote Network Monitoring (RMON)</b>	<b>1133</b>
RMON Standards .....	1134
RMON MIB Hierarchy and Object Groups .....	1134
RMON Alarms, Events, and Statistics .....	1136
<b>PART III-5 TCP/IP APPLICATION LAYER ADDRESSING AND APPLICATION CATEGORIES</b>	<b>1137</b>
<b>70 TCP/IP Application Layer Addressing: Uniform Resource Identifiers, Locators, and Names (URLs, URLs, and URNs)</b>	<b>1139</b>
URI Overview and Standards .....	1140
URI Categories: URLs and URNs .....	1141
URI Standards .....	1142
URL General Syntax .....	1142
Common Internet Scheme Syntax .....	1143
Omission of URL Syntax Elements .....	1144
URL Fragments .....	1145
Unsafe Characters and Special Encodings .....	1145
URL Schemes and Scheme-Specific Syntaxes .....	1146
World Wide Web/Hypertext Transfer Protocol Syntax (http) .....	1146
File Transfer Protocol Syntax (ftp) .....	1147
Electronic Mail Syntax (mailto) .....	1147
Gopher Protocol Syntax (gopher) .....	1148
Network News/Usenet Syntax (news) .....	1148
Network News Transfer Protocol Syntax (nntp) .....	1148
Telnet Syntax (telnet) .....	1149
Local File Syntax (file) .....	1149
Special Syntax Rules .....	1149
URL Relative Syntax and Base URLs .....	1150
Interpretation Rules for Relative URLs .....	1151
Practical Interpretation of Relative URLs .....	1152
URL Length and Complexity Issues .....	1154
URL Wrapping and Delimiting .....	1155
Explicit URL Delimiting and Redirectors .....	1156
URL Abbreviation .....	1156
URL Obscuration, Obfuscation, and General Trickery .....	1156
URNs .....	1159
The Problem with URLs .....	1159
Overview of URNs .....	1160
URN Namespaces and Syntax .....	1160
URN Resolution and Implementation Difficulties .....	1161
<b>71 File and Message Transfer Overview and Application Categories</b>	<b>1163</b>
File Concepts .....	1164
Application Categories .....	1164
General File Transfer Applications .....	1164
Message Transfer Applications .....	1164
The Merging of File and Message Transfer Methods .....	1165

<b>72 File Transfer Protocol (FTP)</b>	<b>1169</b>
FTP Overview, History, and Standards .....	1170
FTP Development and Standardization .....	1170
Overview of FTP Operation .....	1171
FTP Operational Model, Protocol Components, and Key Terminology .....	1172
The Server-FTP Process and User-FTP Process .....	1172
FTP Control Connection and Data Connection .....	1172
FTP Process Components and Terminology .....	1173
Server-FTP Process Components .....	1173
User-FTP Process Components .....	1174
Third-Party File Transfer (Proxy FTP) .....	1174
FTP Control Connection Establishment, User Authentication, and Anonymous FTP Access .....	1175
FTP Login Sequence and Authentication .....	1175
FTP Security Extensions .....	1176
Anonymous FTP .....	1177
FTP Data Connection Management .....	1177
Normal (Active) Data Connections .....	1178
Passive Data Connections .....	1178
Efficiency and Security Issues Related to the Connection Methods .....	1179
FTP General Data Communication and Transmission Modes .....	1181
Stream Mode .....	1181
Block Mode .....	1182
Compressed Mode .....	1182
FTP Data Representation: Data Types, Format Control, and Data Structures .....	1182
FTP Data Types .....	1183
ASCII Data Type Line-Delimiting Issues .....	1184
FTP Format Control .....	1184
FTP Data Structures .....	1185
FTP Internal Command Groups and Protocol Commands .....	1185
FTP Command Groups and Commands .....	1185
FTP Replies .....	1188
Advantages of Using Both Text and Numeric Replies .....	1188
Reply Code Structure and Digit Interpretation .....	1188
FTP Multiple-Line Text Replies .....	1192
FTP User Interface and User Commands .....	1193
Command-Line and Graphical FTP Interfaces .....	1193
Typical FTP User Commands .....	1194
Sample FTP Session .....	1196
<b>73 Trivial File Transfer Protocol (TFTP)</b>	<b>1199</b>
TFTP Overview, History, and Standards .....	1200
Why TFTP Was Needed .....	1200
Comparing FTP and TFTP .....	1201
Overview of TFTP Operation .....	1201
TFTP General Operation, Connection Establishment, and Client/Server Communication .....	1202
Connection Establishment and Identification .....	1203
Lock-Step Client/Server Messaging .....	1203
Difficulties with TFTP's Simplified Messaging Mechanism .....	1204
TFTP Detailed Operation and Messaging .....	1204
Initial Message Exchange .....	1204
Data Block Numbering .....	1205
TFTP Read Process Steps .....	1205
TFTP Write Process Steps .....	1206

TFTP Options and Option Negotiation .....	1208
TFTP Option Negotiation Process .....	1208
TFTP Options .....	1211
TFTP Message Formats .....	1211
Read Request and Write Request Messages .....	1211
Data Messages .....	1212
Acknowledgment Messages .....	1213
Error Messages .....	1213
Option Acknowledgment Messages .....	1214

## **PART III-7 TCP/IP ELECTRONIC MAIL SYSTEM: CONCEPTS AND PROTOCOLS** 1215

<b>74 TCP/IP Electronic Mail System Overview and Concepts</b> <span style="float: right;">1217</span>	
TCP/IP Electronic Mail System Overview and History .....	1218
The Early Days of Email .....	1218
History of TCP/IP Email .....	1219
Overview of the TCP/IP Email System .....	1219
TCP/IP Email Communication Overview .....	1220
TCP/IP Email Message Communication Model .....	1222
Protocol Roles in Email Communication .....	1224

<b>75 TCP/IP Electronic Mail Addresses and Addressing</b> <span style="float: right;">1225</span>	
TCP/IP Email Addressing and Address Resolution .....	1226
Standard DNS-Based Email Addresses .....	1226
Special Requirements of Email Addresses .....	1227
TCP/IP Historical and Special Email Addressing .....	1228
FidoNet Addressing .....	1228
UUCP-Style Addressing .....	1229
Addressing for Gateways .....	1229
TCP/IP Email Aliases and Address Books .....	1230
Multiple Recipient Addressing .....	1230
Mailing Lists .....	1231

## **76 TCP/IP Electronic Mail Message Formats and Message Processing: RFC 822 and MIME** 1233

TCP/IP Email RFC 822 Standard Message Format Overview .....	1234
Development of the RFC 822 Message Format Standard .....	1235
Overview of RFC 822 Messages .....	1235
General RFC 822 Message Structure .....	1236
TCP/IP Email RFC 822 Standard Message Format Header Fields and Groups .....	1237
Header Field Structure .....	1237
Header Field Groups .....	1237
Common Header Field Groups and Header Fields .....	1238
TCP/IP Email RFC 822 Standard Message Format Processing and Interpretation .....	1241
MIME Overview .....	1242
MIME Capabilities .....	1243
MIME Standards .....	1244
MIME Basic Structures and Headers .....	1245
Basic Structures .....	1246
MIME Entities .....	1246
Primary MIME Headers .....	1246
Additional MIME Headers .....	1247

MIME Content-Type Header and Discrete Media .....	1248
Content-Type Header Syntax .....	1248
Discrete Media Types and Subtypes .....	1249
MIME Composite Media Types: Multipart and Encapsulated Message Structures .....	1253
MIME Multipart Message Type .....	1253
Multipart Message Encoding .....	1254
MIME Encapsulated Message Type .....	1257
MIME Content-Transfer-Encoding Header and Encoding Methods .....	1257
7-Bit and 8-Bit Encoding .....	1258
Quoted-Printable Encoding .....	1258
Base64 Encoding .....	1258
MIME Extension for non-ASCII Mail Message Headers .....	1261

## **77 TCP/IP Electronic Mail Delivery Protocol: The Simple Mail Transfer Protocol (SMTP) 1263**

SMTP Overview, History, and Standards .....	1264
SMTP Standards .....	1264
SMTP Communication and Message Transport Methods .....	1265
Terminology: Client/Server and Sender/Receiver .....	1267
SMTP Connection and Session Establishment and Termination .....	1267
Overview of Connection Establishment and Termination .....	1268
Connection Establishment and Greeting Exchange .....	1268
Connection Establishment Using SMTP Extensions .....	1269
Connection Termination .....	1270
SMTP Mail Transaction Process .....	1271
Overview of SMTP Mail Transaction .....	1271
SMTP Mail Transaction Details .....	1272
SMTP Special Features, Capabilities, and Extensions .....	1274
SMTP Special Features and Capabilities .....	1275
SMTP Extensions .....	1276
SMTP Security Issues .....	1277
SMTP Commands .....	1279
SMTP Replies and Reply Codes .....	1281
Reply Code Structure and Digit Interpretation .....	1281
SMTP Multiple-Line Text Replies .....	1284
Enhanced Status Code Replies .....	1284

## **78 TCP/IP Electronic Mail Access and Retrieval Protocols and Methods 1285**

TCP/IP Email Mailbox Access Model, Method, and Protocol Overview .....	1286
Email Access and Retrieval Models .....	1287
TCP/IP Post Office Protocol (POP/POP3) .....	1288
POP Overview, History, Versions, and Standards .....	1288
POP3 General Operation .....	1290
POP3 Session States .....	1290
TCP/IP Internet Message Access Protocol (IMAP/IMAP4) .....	1297
IMAP Overview, History, Versions, and Standards .....	1298
IMAP General Operation .....	1300
IMAP Session States .....	1300
IMAP Commands, Results, and Responses .....	1302
IMAP Not Authenticated State: User Authentication Process and Commands .....	1306
IMAP Authenticated State: Mailbox Manipulation/Selection Process and Commands .....	1307
IMAP Selected State: Message Manipulation Process and Commands .....	1309
TCP/IP Direct Server Email Access .....	1311
TCP/IP World Wide Web Email Access .....	1313

**79 World Wide Web and Hypertext Overview and Concepts 1317**

World Wide Web and Hypertext Overview and History .....	1318
History of Hypertext .....	1318
The World Wide Web Today .....	1319
World Wide Web System Concepts and Components .....	1320
Major Functional Components of the Web .....	1320
Web Servers and Web Browsers .....	1321
World Wide Web Media and the Hypertext Markup Language .....	1322
Overview of HTML .....	1322
HTML Elements and Tags .....	1323
Common HTML Elements .....	1324
Common Text Formatting Tags .....	1326
World Wide Web Addressing: HTTP Uniform Resource Locators .....	1326
HTTP URL Syntax .....	1327
Resource Paths and Directory Listings .....	1328

**80 HTTP General Operation and Connections 1329**

HTTP Versions and Standards .....	1330
HTTP/0.9 .....	1330
HTTP/1.0 .....	1330
HTTP/1.1 .....	1331
Future HTTP Versions .....	1332
HTTP Operational Model and Client/Server Communication .....	1333
Basic HTTP Client/Server Communication .....	1333
Intermediaries and the HTTP Request/Response Chain .....	1334
The Impact of Caching on HTTP Communication .....	1335
HTTP Transitory and Persistent Connections and Pipelining .....	1336
Persistent Connections .....	1337
Pipelining .....	1337
HTTP Persistent Connection Establishment and Management .....	1338

**81 HTTP Messages, Methods, and Status Codes 1341**

HTTP Generic Message Format .....	1342
HTTP Request Message Format .....	1343
Request Line .....	1344
Headers .....	1346
HTTP Response Message Format .....	1346
Status Line .....	1347
Headers .....	1348
HTTP Methods .....	1349
Common Methods .....	1349
Other Methods .....	1350
Safe and Idempotent Methods .....	1351
HTTP Status Codes and Reason Phrases .....	1352
Status Code Format .....	1352
Reason Phrases .....	1353
The 100 (Continue) Preliminary Reply .....	1356

<b>82 HTTP Message Headers</b>	<b>1357</b>
HTTP General Headers .....	1358
Cache-Control Headers .....	1358
Warning .....	1359
Other HTTP General Headers .....	1360
HTTP Request Headers .....	1361
HTTP Response Headers .....	1364
HTTP Entity Headers .....	1365
<b>83 HTTP Entities, Transfers, Coding Methods, and Content Management</b>	<b>1369</b>
HTTP Entities and Internet Media Types .....	1370
Media Types and Subtypes .....	1370
HTTP's Use of Media Types .....	1371
Differences in HTTP and MIME Constructs .....	1371
HTTP Content and Transfer Encodings .....	1372
HTTP's Two-Level Encoding Scheme .....	1373
Use of Content and Transfer Encodings .....	1373
HTTP Data Length Issues, Chunked Transfers, and Message Trailers .....	1374
Dynamic Data Length .....	1375
Chunked Transfers and Message Trailers .....	1375
HTTP Content Negotiation and Quality Values .....	1378
Content Negotiation Techniques .....	1378
Quality Values for Preference Weights .....	1380
<b>84 HTTP Features, Capabilities, and Issues</b>	<b>1381</b>
HTTP Caching Features and Issues .....	1382
Benefits of HTTP Caching .....	1382
Cache Locations .....	1383
Cache Control .....	1384
Important Caching Issues .....	1385
HTTP Proxy Servers and Proxying .....	1386
Benefits of Proxies .....	1386
Comparing Proxies and Caches .....	1387
Important Proxying Issues .....	1387
HTTP Security and Privacy .....	1388
HTTP Authentication Methods .....	1389
Security and Privacy Concerns and Issues .....	1389
Methods for Ensuring Privacy in HTTP .....	1390
HTTP State Management Using Cookies .....	1390
Issues with Cookies .....	1391
Managing Cookie Use .....	1393
<b>PART III-9 OTHER FILE AND MESSAGE TRANSFER APPLICATIONS</b>	<b>1395</b>
<b>85 Usenet (Network News) and the TCP/IP Network News Transfer Protocol (NNTP)</b>	<b>1397</b>
Usenet Overview, History, and Operation .....	1398
History of Usenet .....	1398
Usenet Operation and Characteristics .....	1399
Usenet Transport Methods .....	1400
Usenet Communication Model .....	1401
Usenet's Public Distribution Orientation .....	1402
Usenet Communication Process .....	1402

Message Propagation and Server Organization .....	1404
Usenet Addressing: Newsgroups .....	1404
Usenet Message Format and Special Headers .....	1408
Usenet Header Categories and Common Headers .....	1408
Additional Usenet Headers .....	1410
Usenet MIME Messages .....	1411
NNTP Overview and General Operation .....	1411
NNTP Interserver Communication Process: News Article Propagation .....	1413
The Usenet Server Structure .....	1413
Basic NNTP Propagation Methods .....	1414
NNTP Client-Server Communication Process: News Posting and Access .....	1416
News Posting, Access, and Reading .....	1417
News Access Methods .....	1418
Other Client/Server Functions .....	1418
Article Threading .....	1419
NNTP Commands and Command Extensions .....	1420
Command Syntax .....	1420
Base Command Set .....	1420
NNTP Command Extensions .....	1422
NNTP Status Responses and Response Codes .....	1426

<b>86 Gopher Protocol (Gopher)</b>	<b>1431</b>
Gopher Overview and General Operation .....	1431
Information Storage on Gopher Servers .....	1432
Gopher Client/Server Operation .....	1432
Important Differences Between Gopher and the Web .....	1433
Gopher's Role in the Modern Internet .....	1433

## **PART III-10 INTERACTIVE AND ADMINISTRATIVE UTILITIES AND PROTOCOLS**

<b>87 TCP/IP Interactive and Remote Application Protocols</b>	<b>1437</b>
Telnet Protocol .....	1438
Telnet Overview, History, and Standards .....	1438
Telnet Connections and Client/Server Operation .....	1441
Telnet Communications Model and the Network Virtual Terminal (NVT) .....	1443
Telnet Protocol Commands .....	1446
Telnet Interrupt Handling .....	1449
Telnet Options and Option Negotiation .....	1450
Berkeley Remote (r) Commands .....	1454
Berkeley Remote Login (rlogin) .....	1455
Berkeley Remote Shell (rsh) .....	1456
Other Berkeley Remote Commands .....	1457
Internet Relay Chat Protocol (IRC) .....	1458
IRC Communication Model and Client/Server Operation .....	1459
Messaging and IRC Channels .....	1459
IRC and the Modern Internet .....	1460

<b>88 TCP/IP Administration and Troubleshooting Utilities and Protocols</b>	<b>1461</b>
TCP/IP Host Name Utility (hostname) .....	1462
TCP/IP Communication Verification Utility (ping) .....	1463
Operation of the ping Utility .....	1464
Basic Use of ping .....	1464

Methods of Diagnosing Connectivity Problems Using ping .....	1465
ping Options and Parameters .....	1466
TCP/IP Route Tracing Utility (traceroute) .....	1467
Operation of the traceroute Utility .....	1468
Basic Use of the traceroute Utility .....	1469
traceroute Options and Parameters .....	1470
TCP/IP Address Resolution Protocol Utility (arp) .....	1471
TCP/IP DNS Name Resolution and Lookup Utilities (nslookup, host, and dig) .....	1472
The nslookup Utility .....	1473
The host Utility .....	1475
The dig Utility .....	1476
TCP/IP DNS Registry Database Lookup Utility (whois/nicname) .....	1477
TCP/IP Network Status Utility (netstat) .....	1479
The UNIX netstat Utility .....	1480
The Windows netstat Utility .....	1482
TCP/IP Configuration Utilities (ifconfig, ipconfig, and winipcfg) .....	1484
The ifconfig Utility for UNIX .....	1484
The ipconfig for Windows NT, 2000, and XP .....	1486
The winipcfg Utility for Windows 95, 98, and Me .....	1488
Miscellaneous TCP/IP Troubleshooting Protocols .....	1489

**INDEX** **1491**

**RFCs BY NUMBER** **1537**

## **L I S T   O F   F I G U R E S**

	page
Address resolution problems with large hardware address size	Figure 13-3 208
Address Resolution Protocol (ARP) transaction process	Figure 13-5 215
Address resolution through direct mapping	Figure 13-2 207
Address resolution, why necessary	Figure 13-1 205
ARP message format	Figure 13-6 218
ARP Proxy operation	Figure 13-7 222
ARP proxying by Mobile IP home agent	Figure 30-9 500
BGP general message format	Figure 40-2 665
BGP Keepalive message format	Figure 40-5 673
BGP Notification message format	Figure 40-6 674
BGP Open message format	Figure 40-3 668
BGP topology and designations, sample	Figure 40-1 653
BGP Update message format	Figure 40-4 670
Binary, decimal, and hexadecimal representations of IPv6 addresses	Figure 25-2 378
Binary information representations and terms	Figure 4-1 64
Binary, octal, and hexadecimal number representations	Figure 4-2 67
Bit mask, AND, clearing bits using	Figure 4-3 77
BOOTP, general operation of	Figure 60-1 982
BOOTP message format	Figure 60-3 988
BOOTP operation	Figure 60-2 985
BOOTP operation using a relay agent	Figure 60-5 994
BOOTP Vendor-Specific Area format showing vendor information fields	Figure 60-4 990
CIDR (/15) address block, hierarchical division of, example of	Figure 20-3 325
CIDR (slash) notation and its subnet mask equivalent	Figure 20-2 320
Circuit switching	Figure 1-1 14
Class A, Class B, and Class C networks, default subnet masks for	Figure 18-4 282

	page
Class B network, determining subnet addresses for	Figure 19-7 308
Class B network, determining the custom subnet mask for	Figure 19-5 304
Class C (/24) network split into eight conventional subnets	Figure 18-6 293
Class C (/24) network split using VLSM	Figure 18-7 294
Class C network, determining host addresses for	Figure 19-8 312
Class C network, determining subnet addresses for	Figure 19-6 307
Class C network, determining the custom subnet mask for	Figure 19-4 304
Class C networks, custom subnet masks for	Figure 18-5 285
Class determination algorithm for classful IP addresses	Figure 17-2 259
Classful addressing, main problem with	Figure 17-5 271
Classless addressing (CIDR) solves the granularity problem	Figure 20-1 318
Client-server networking	Figure 1-6 25
 DHCP Automatic Private IP Addressing (APIPA)	 Figure 64-1 1060
DHCP client finite state machine	Figure 62-1 1020
DHCP lease allocation process	Figure 62-2 1025
DHCP lease reallocation process	Figure 62-3 1028
DHCP lease renewal and rebinding processes	Figure 62-4 1032
DHCP life cycle example	Figure 61-1 1007
DHCP message format	Figure 63-1 1039
DHCP multiple-server non-overlapping scopes	Figure 61-3 1011
DHCP Options field format	Figure 63-2 1042
DHCP parameter configuration process	Figure 62-5 1034
DHCP scope	Figure 61-2 1010
DNS common RR format	Figure 57-4 937
DNS functions	Figure 52-1 853
DNS general message format	Figure 57-1 931
DNS IN-ADDR.ARPA reverse name resolution hierarchy	Figure 56-4 922
DNS labels and domain name construction	Figure 53-4 864
DNS message header format	Figure 57-2 934
DNS message Question section format	Figure 57-3 935
DNS name resolution, iterative	Figure 56-1 913
DNS name resolution process, example of	Figure 56-3 919
DNS name resolution, recursive	Figure 56-2 914
DNS name space “family tree”	Figure 53-3 862
DNS root name servers (Internet), geographic locations of	Figure 55-2 901
DNS RR master file and binary field formats	Figure 55-1 891

	page
DNS standard name notation	Figure 57-6
DNS Start Of Authority (SOA) RR data format	941
DNS tree-related and domain-related terminology	Figure 57-5
DNS zones of authority	861
Dynamic address resolution	Figure 54-2
	883
Email communication model	Figure 13-4
	210
	Figure 74-1
	1223
Flat name architecture (flat name space)	Figure 50-4
FTP active data connection	832
FTP connection establishment and user authentication	Figure 72-3
FTP operational model	1179
FTP passive data connection	Figure 72-2
	1176
FTP reply code format	Figure 72-1
	1173
Global hierarchical domain architecture, example of	Figure 72-4
Global object name hierarchy and SNMP MIB hierarchies	Figure 72-5
	1180
	Figure 53-1
	859
Hierarchical address division using CIDR	Figure 66-2
Hierarchical name architecture (structured name space)	Figure 20-4
HTTP client/server communication	833
HTTP Request message format	Figure 80-1
HTTP request/response chain, impact of caching on	Figure 81-1
HTTP request/response chain using intermediaries	Figure 84-1
HTTP Response message format	1334
	1335
	Figure 80-2
	1347
	Figure 81-2
ICMP common message format	Figure 81-1
ICMP general operation	519
ICMP Redirect message, host redirection using	Figure 31-1
ICMPv4 Address Mask Request and Address Mask Reply message format	Figure 32-5
ICMPv4 Destination Unreachable message format	Figure 33-5
ICMPv4 Echo and Echo Reply message format	531
ICMPv4 Parameter Problem message format	Figure 32-1
ICMPv4 Redirect message format	Figure 33-1
ICMPv4 Router Advertisement Message format	523
ICMPv4 Router Solicitation Message format	Figure 33-3
ICMPv4 Source Quench message format	Figure 33-4
	537
	541
	542
	533
	Figure 32-6
	526
	Figure 32-2

	page
ICMPv4 Time Exceeded message format	529
ICMPv4 Timestamp and Timestamp Reply message format	539
ICMPv4 Traceroute message format	546
ICMPv6 Destination Unreachable message format	548
ICMPv6 Echo Request and Echo Reply message format	559
ICMPv6 MTU option format	574
ICMPv6 Neighbor Advertisement message format	564
ICMPv6 Neighbor Solicitation message format	565
ICMPv6 Packet Too Big message format	551
ICMPv6 Parameter Problem message format	555
ICMPv6 Prefix Information option format	573
ICMPv6 Redirect message format	567
ICMPv6 Redirected Header option format	574
ICMPv6 Router Advertisement message format	560
ICMPv6 Router Renumbering message format	569
ICMPv6 Router Solicitation message format	562
ICMPv6 Source Link-Layer Address option format	571
ICMPv6 Target Link-Layer Address option format	572
ICMPv6 Time Exceeded message format	554
IEEE 802 MAC addresses, converting to IPv6 modified EUI-64 identifiers	390
IMAP FSM	1301
Internet DNS organizational (generic) TLDs	872
Internet standards organizations	54
Internetwork access with a name system	828
Internetwork access without a name system	827
IP address binary, hexadecimal, and dotted decimal representations	246
IP address class bit assignments and network/host ID sizes	261
IP address division (basic): network ID and host ID	248
IP address division, mid-octet	249
IP address subnet ID, determining through subnet masking	280
IP addresses (multicast), mapping to IEEE 802 multicast MAC addresses	224
IP datagram encapsulation	331
IP datagram, expiration of, and Time Exceeded message generation	528
IP datagram next-hop routing	356
IP datagrams, direct and indirect (routed) delivery of	352
IP interfaces for common network devices	243
IP maximum transmission unit (MTU) and fragmentation	341

	page
IP Multicast address ranges and uses	268
IP Network Address Translation (NAT) terminology	432
IP routing and routing tables	358
IP: internetwork datagram delivery, main function of	236
IPsec Authentication Header (AH) format	465
IPsec bump in the stack (BITS) architecture	455
IPsec bump in the wire (BITW) architecture	456
IPsec ESP format	471
IPsec protocols and components, overview	453
IPsec transport mode operation	458
IPsec tunnel mode operation	459
IPv4 address space, division into classes	257
IPv4 and IPv6 address space sizes, relative, a (poor) representation of	377
IPv4 datagram format	334
IPv4 datagram format with IPsec AH	464
IPv4 datagram format with IPsec ESP	469
IPv4 datagram fragmentation	342
IPv4 datagram fragmentation process	345
IPv4 Options field format	337
IPv4-compatible embedded IPv6 address representation	393
IPv4-mapped embedded IPv6 address representation	393
IPv6 datagram format with IPsec Authentication Header (AH)	463
IPv6 datagram format with IPsec ESP	468
IPv6 datagram fragmentation	419
IPv6 extension header linking using the Next Header field	408
IPv6 Fragment extension header format	411
IPv6 general datagram structure	403
IPv6 global unicast address format	385
IPv6 Hop-By-Hop Options and Destination Options header formats	413
IPv6 main header format	405
IPv6 multicast address format	394
IPv6 multicast scope	396
IPv6 Routing extension header format	410
IPv6 solicited-node address calculation	397
IPv6 unicast routing prefix structure, example of	387
Message routing in the OSI Reference Model	99

	page	
MIME base64 encoding	Figure 76-2	1260
MIME multipart message structure	Figure 76-1	1256
Mobile devices on IP internetworks, main problem with	Figure 30-1	477
Mobile IP encapsulation and tunneling	Figure 30-8	497
Mobile IP, general operation of	Figure 30-2	481
Mobile IP inefficiency worst-case scenario	Figure 30-10	502
Mobile IP Mobility Agent Advertisement Extension format	Figure 30-4	489
Mobile IP operation with a foreign agent care-of address	Figure 30-3	484
Mobile IP Prefix-Lengths Extension format	Figure 30-5	490
Mobile IP Registration Reply Message format	Figure 30-7	495
Mobile IP Registration Request message format	Figure 30-6	494
Multihomed devices on an IP internetwork	Figure 16-5	253
Multilink PPP architecture	Figure 11-1	176
 Name system functions	 Figure 50-3	 830
NAT, bidirectional (two-way/inbound), operation of	Figure 28-3	438
NAT (Overlapping NAT/Twice NAT), operation of	Figure 28-5	446
NAT, Port-Based (overloaded), operation of	Figure 28-4	441
NAT, unidirectional (traditional/outbound), operation of	Figure 28-2	436
ND host redirection using an ICMPv6 Redirect message	Figure 36-2	587
Neighbor Discovery (ND) protocol functional groups and functions	Figure 36-1	578
Network message formatting	Figure 1-3	19
NFS architectural components	Figure 58-1	958
NNTP article propagation using the push model	Figure 85-3	1415
NNTP full newsgroup retrieval process	Figure 85-4	1419
 OSI Reference Model and TCP/IP model layers	 Figure 8-2	 129
OSI Reference Model data encapsulation	Figure 5-5	96
OSI Reference Model interfaces for vertical communication	Figure 5-3	92
OSI Reference Model layer relationships and terminology	Figure 5-2	91
OSI Reference Model layers	Figure 5-1	89
OSI Reference Model mnemonics	Figure 7-1	116
OSI Reference Model PDU and SDU encapsulation	Figure 5-6	97
OSI Reference Model protocols: horizontal communication	Figure 5-4	94
OSPF AS, sample	Figure 39-1	629
OSPF AS with Costs, sample	Figure 39-3	634
OSPF calculated SPF tree	Figure 39-5	637

	page
OSPF common header format	640
OSPF Database Description message format	642
OSPF Hello message format	641
OSPF hierarchical topology AS, sample	632
OSPF Link State Acknowledgment message format	644
OSPF Link State Advertisement header format	645
OSPF Link State Request Message format	643
OSPF Link State Update message format	644
OSPF route determination using the SPF algorithm	635
 Packet switching	 14
PAP authentication	163
PAR, enhanced	735
Peer-to-peer networking	24
POP3 finite state machine	1291
POP3 mail exchange process	1296
POP3 user authentication process	1292
PPP Challenge Handshake Authentication Protocol (CHAP) authentication	164
PPP CHAP Challenge and Response frame format	195
PPP CHAP Success and Failure frame format	195
PPP control message carrying options	189
PPP control message format	187
PPP control message option format	189
PPP data frame, sample	184
PPP general frame format	183
PPP IP Control Protocol (IPCP) message exchanges	161
PPP LCP link configuration process	157
PPP Link Control Protocol (LCP) message exchanges	156
PPP location in the TCP/IP architecture	145
PPP MP fragmentation	199
PPP MP long fragment frame format	197
PPP MP short fragment frame format	198
PPP operation, overview	148
PPP PAP Authenticate-Ack and Authenticate-Nak frame format	193
PPP PAP Authenticate-Request frame format	193
PPP phases	151
Protocol, unreliable, operation of	732

	page
Reliability (basic): positive acknowledgment with retransmission (PAR)	Figure 46-3 733
Reverse Address Resolution Protocol (RARP) operation	Figure 14-2 229
Reverse Address Resolution Protocol (RARP), operation of	Figure 14-1 229
RIP AS, sample	Figure 38-1 601
RIP counting to infinity problem	Figure 38-3 608
RIP problem solving using split horizon with poisoned reverse	Figure 38-4 613
RIP, propagation of network routing information using	Figure 38-2 603
RIP-1 message format	Figure 38-5 616
RIP-2 message format	Figure 38-6 620
RIPng message format	Figure 38-7 622
Router loop (example)	Figure 34-3 553
Serial Line Internet Protocol (SLIP), operation of	Figure 9-1 143
SMTP mail transaction process	Figure 77-2 1273
SMTP transaction session establishment and termination	Figure 77-1 1269
SNMP general message format	Figure 68-1 1118
SNMP information poll process	Figure 67-1 1105
SNMP management information base (MIB)	Figure 66-1 1089
SNMP object modification process	Figure 67-2 1108
SNMP operational model	Figure 65-1 1074
SNMP Remote Network Monitoring (RMON) MIB hierarchy	Figure 69-1 1135
SNMPv1 common PDU format	Figure 68-3 1121
SNMPv1 general message format	Figure 68-2 1119
SNMPv1 Trap-PDU format	Figure 68-4 1122
SNMPv2 common PDU format	Figure 68-8 1128
SNMPv2 GetBulkRequest-PDU format	Figure 68-9 1128
SNMPv2c general message format	Figure 68-6 1124
SNMPv2p general message format	Figure 68-5 1123
SNMPv2u general message format	Figure 68-7 1125
SNMPv3 general message format	Figure 68-10 1130
Subnetted network, determining the subnet mask of	Figure 18-2 279
Subnetting, Class B, example of	Figure 19-3 302
Subnetting Class B network	Figure 18-1 277
Subnetting, Class C, example of	Figure 19-2 301
Subnetting design trade-off for Class C networks	Figure 19-1 300
TCP aggressive retransmission example	Figure 49-2 800

	page
TCP connection termination procedure	Figure 47-5 765
TCP data stream processing and segment packaging	Figure 46-1 730
TCP finite state machine (FSM)	Figure 47-1 750
TCP header checksum calculation	Figure 48-3 776
TCP pseudo header for checksum calculation	Figure 48-2 775
TCP receive categories and pointers	Figure 48-5 783
TCP retransmission with selective acknowledgment (SACK)	Figure 49-3 802
TCP segment format	Figure 48-1 772
TCP send window, sliding the	Figure 46-8 739
TCP sequence number synchronization	Figure 47-4 759
TCP silly window syndrome (SWS)	Figure 49-6 814
TCP simultaneous connection termination procedure	Figure 47-6 766
TCP simultaneous open connection establishment procedure	Figure 47-3 756
TCP stream categories and window after sending usable window bytes	Figure 46-7 738
TCP three-way handshake connection establishment procedure	Figure 47-2 755
TCP transaction example showing client's send pointers	Figure 48-7 788
TCP transaction example showing the server's send pointers	Figure 48-6 787
TCP transaction example with retransmission	Figure 49-1 797
TCP transmission categories, send window, and pointers	Figure 48-4 782
TCP transmission stream categories and send window terminology	Figure 46-6 738
TCP transmission stream, conceptual division into categories	Figure 46-5 736
TCP window, problem with shrinking	Figure 49-5 810
TCP window size adjustments and flow control	Figure 49-4 807
TCP/IP autonomous system (AS) routing architecture	Figure 37-1 594
TCP/IP client/server application port mechanics	Figure 43-3 705
TCP/IP client/server operation	Figure 8-1 126
TCP/IP Internet Standard Management Framework, components	Figure 65-2 1076
TCP/IP, process multiplexing and demultiplexing in	Figure 43-1 698
TCP/IP process multiplexing/demultiplexing using TCP/UDP ports	Figure 43-2 700
TCP/IP protocols	Figure 8-3 134
Telnet communication and the Network Virtual Terminal (NVT)	Figure 87-1 1444
TFTP acknowledgment message format	Figure 73-6 1213
TFTP data message format	Figure 73-5 1213
TFTP error message format	Figure 73-7 1213
TFTP OACK message format	Figure 73-8 1214
TFTP read process	Figure 73-1 1206
TFTP read process with option negotiation	Figure 73-3 1210



	page	
TFTP RRQ/WRQ message format	Figure 73-4	1212
TFTP write process	Figure 73-2	1207
traceroute/tracert utility, operation of	Figure 88-1	1469
UDP message format	Figure 44-1	714
UDP pseudo header format	Figure 44-2	715
Unicast, multicast, and broadcast message addressing and transmission	Figure 1-4	21
Uniform Resource Locator (URL), example of	Figure 70-1	1144
Usenet (network news) communication model	Figure 85-1	1403
Usenet newsgroup hierarchies	Figure 85-2	1406
VLSM example	Figure 18-8	295
Windows 95/98/Me winipcfg utility	Figure 88-2	1489
World Wide Web, major functional components of	Figure 79-1	1321

## L I S T   O F   T A B L E S

	page
ARP Hardware Type (HRD) Field Values	Table 13-2 217
ARP Message Format	Table 13-1 217
ARP Opcode (OP) Field Values	Table 13-3 218
BGP General Message Format	Table 40-4 664
BGP Keepalive Message Format	Table 40-11 673
BGP Notification Message Error Codes	Table 40-13 675
BGP Notification Message Error Subcodes	Table 40-14 675
BGP Notification Message Format	Table 40-12 674
BGP Open Message Format	Table 40-5 666
BGP Open Message Optional Parameters	Table 40-6 667
BGP Path Attributes, Summary of	Table 40-3 659
BGP Update Message Attribute Flags	Table 40-9 671
BGP Update Message Attribute Type Codes	Table 40-10 671
BGP Update Message Format	Table 40-7 669
BGP Update Message Path Attributes	Table 40-8 669
BGP Versions and Defining Standards	Table 40-1 649
BGP-4, Additional Defining Standards for	Table 40-2 650
Binary Addition	Table 4-7 72
Binary and Decimal Number Equivalents	Table 4-2 66
Binary Information Group Representations and Terms	Table 4-1 63
Bit Mask, AND, Clearing Bits Using	Table 4-15 77
Bit Mask, OR, Setting Bits Using	Table 4-14 76
Bit Mask, XOR, Inverting Bits Using	Table 4-16 77
BOOTP Message Format	Table 60-1 986
BOOTP Message HType Values	Table 60-2 987
BOOTP Vendor Information Field Format	Table 60-3 990
CHAP Challenge and Response Frame Subfields	Table 12-11 194
CIDR Address Blocks and Classful Address Equivalents	Table 20-1 323

		page
Conversion, Binary, Octal, and Hexadecimal Digits	Table 4-3	68
Conversion, Decimal to Binary Numbers	Table 4-5	70
Conversion, Decimal to Hexadecimal Numbers	Table 4-6	71
Conversion, Hexadecimal to Decimal Numbers	Table 4-4	69
DHCP Client Finite State Machine	Table 62-1	1018
DHCP Message Format	Table 63-1	1039
DHCP Message HTYPE Values	Table 63-2	1041
DHCP Message Type (Option 53) Values	Table 63-13	1051
DHCP Option Categories	Table 63-4	1043
DHCP Option Format	Table 63-3	1043
DHCP Options: DHCP Extensions	Table 63-12	1050
DHCP/BOOTP Options: Application and Service Parameters	Table 63-10	1049
DHCP/BOOTP Options: IP Layer Parameters per Host	Table 63-6	1047
DHCP/BOOTP Options: IP Layer Parameters per Interface	Table 63-7	1047
DHCP/BOOTP Options: Link Layer Parameters per Interface	Table 63-8	1048
DHCP/BOOTP Options: RFC 1497 Vendor Extensions	Table 63-5	1045
DHCP/BOOTP Options: TCP Parameters	Table 63-9	1048
DNS Address RR Data Format	Table 57-8	937
DNS Canonical Name RR Data Format	Table 57-10	938
DNS Common Resource Record Format	Table 57-7	936
DNS General Message Format	Table 57-1	930
DNS Mail Exchange RR Data Format	Table 57-13	939
DNS Message Header Format	Table 57-2	933
DNS Message Question Section Format	Table 57-5	935
DNS Name Server RR Data Format	Table 57-9	937
DNS Pointer RR Data Format	Table 57-12	938
DNS Resource Records (Common), Summary of	Table 55-1	892
DNS Start Of Authority RR Data Format	Table 57-11	938
DNS Text RR Data Format	Table 57-14	940
FTP Access Control Commands	Table 72-1	1186
FTP Protocol Service Commands	Table 72-3	1187
FTP Reply Code Format: First Digit Interpretation	Table 72-4	1189
FTP Reply Code Format: Second Digit Interpretation	Table 72-5	1189
FTP Reply Codes	Table 72-6	1190
FTP Session, Sample	Table 72-8	1196
FTP Transfer Parameter Commands	Table 72-2	1187
FTP User Commands, Common	Table 72-7	1194

	page
Header OpCode Values	Table 57-3 933
Header RCode Values	Table 57-4 934
Hexadecimal Addition	Table 4-8 72
host Utility Options and Parameters, Typical	Table 88-6 1475
HTML Elements, Common	Table 79-1 1324
HTTP Cache-Control Directives	Table 82-1 1359
HTTP Status Code Format: First-Digit Interpretation	Table 81-1 1353
HTTP Status Codes and Reason Phrases	Table 81-2 1354
HTTP Warning Header Codes	Table 82-2 1360
ICMP Common Message Format	Table 31-3 518
ICMP Message Classes, Types, and Codes	Table 31-2 513
ICMP Redirect Message Interpretation Codes	Table 32-6 532
ICMPv4 Address Mask Request and Address Mask Reply Message Format	Table 33-5 544
ICMPv4 Destination Unreachable Message Format	Table 32-1 522
ICMPv4 Destination Unreachable Message Subtypes	Table 32-2 523
ICMPv4 Echo and Echo Reply Message Format	Table 33-1 536
ICMPv4 Parameter Problem Message Format	Table 32-7 534
ICMPv4 Parameter Problem Message Interpretation Codes	Table 32-8 534
ICMPv4 Redirect Message Format	Table 32-5 531
ICMPv4 Router Advertisement Message Format	Table 33-3 541
ICMPv4 Router Solicitation Message Format	Table 33-4 542
ICMPv4 Source Quench Message Format	Table 32-3 526
ICMPv4 Time Exceeded Message Format	Table 32-4 529
ICMPv4 Timestamp and Timestamp Reply Message Format	Table 33-2 538
ICMPv4 Traceroute Message Format	Table 33-6 545
ICMPv6 Destination Unreachable Message Format	Table 34-1 549
ICMPv6 Destination Unreachable Message Subtypes	Table 34-2 549
ICMPv6 Echo Request and Echo Reply Message Format	Table 35-1 559
ICMPv6 MTU Option Format	Table 35-16 574
ICMPv6 Neighbor Advertisement Message Flags	Table 35-6 564
ICMPv6 Neighbor Advertisement Message Format	Table 35-5 563
ICMPv6 Neighbor Solicitation Message Format	Table 35-7 565
ICMPv6 Packet Too Big Message Format	Table 34-3 551
ICMPv6 Parameter Problem Message Format	Table 34-5 555
ICMPv6 Parameter Problem Message Interpretation Codes	Table 34-6 556
ICMPv6 Prefix Information Option Flags	Table 35-14 573
ICMPv6 Prefix Information Option Format	Table 35-13 572
ICMPv6 Redirect Message Format	Table 35-8 566
ICMPv6 Redirected Header Option Format	Table 35-15 574

	page
ICMPv6 Router Advertisement Message Autoconfiguration Flags	Table 35-3 561
ICMPv6 Router Advertisement Message Format	Table 35-2 561
ICMPv6 Router Renumbering Message Flags	Table 35-10 570
ICMPv6 Router Renumbering Message Format	Table 35-9 569
ICMPv6 Router Solicitation Message Format	Table 35-4 562
ICMPv6 Source Link-Layer Address Option Format	Table 35-11 571
ICMPv6 Target Link-Layer Address Option Format	Table 35-12 572
ICMPv6 Time Exceeded Message Format	Table 34-4 553
ifconfig Interface Configuration Parameters (UNIX), Typical	Table 88-13 1486
ifconfig Syntaxes, Options, and Parameters (UNIX), Typical	Table 88-11 1485
ifconfig Universal Options and Parameters (UNIX), Typical	Table 88-12 1485
IMAP "Any State" Commands	Table 78-2 1304
IMAP Authenticated State Commands	Table 78-4 1308
IMAP Not Authenticated State Commands	Table 78-3 1306
IMAP Selected State Commands	Table 78-5 1310
Internet DNS Organizational (Generic) Top-Level Domains	Table 54-1 872
Internet DNS Root Name Servers	Table 55-2 900
Internet Protocol Version 4 (IPv4) Datagram Format	Table 21-1 332
IP Address Class Bit Patterns, First-Octet Ranges, and Address Ranges	Table 17-2 260
IP Address Class Network and Host Capacities	Table 17-3 262
IP Address Classes and Class Characteristics and Uses	Table 17-1 256
IP Address Patterns with Special Meanings	Table 17-4 264
IP Addresses, Reserved, Private, and Loopback	Table 17-5 267
IP Multicast Address Ranges and Uses	Table 17-6 268
IP Multicast Addresses, Well-Known	Table 17-7 269
IP Security (IPsec) Standards, Important	Table 29-1 451
ipconfig Options and Parameters (Windows), Typical	Table 88-14 1487
IPsec Authentication Header (AH) Format	Table 29-2 465
IPsec Encapsulating Security Payload (ESP) Format	Table 29-3 470
IPv4 Flags Subfields	Table 21-2 333
IPv4 Option Format	Table 21-5 337
IPv4 Options, Common	Table 21-7 338
IPv4 Options: Option Type Subfields	Table 21-6 337
IPv4 Protocol Subfields	Table 21-3 334
IPv4 Type of Service (TOS) Field, Original Definition of	Table 21-4 336
IPv6 Address Space Allocations	Table 25-1 382
IPv6 Extension Headers	Table 26-4 409
IPv6 Fragment Extension Header Format	Table 26-6 411
IPv6 General Datagram Structure	Table 26-1 403
IPv6 Global Unicast Address Format	Table 25-2 384

	page
IPv6 Global Unicast Address Format, Generic	Table 25-3 384
IPv6 Main Header Format	Table 26-2 404
IPv6 Multicast Address Format	Table 25-6 395
IPv6 Next Header Values, Common	Table 26-3 406
IPv6 Option Format	Table 26-7 414
IPv6 Option Type Subfields	Table 26-8 414
IPv6 Routing Extension Header Format	Table 26-5 410
IPv6 Unicast Routing Prefix Structure, Example	Table 25-5 386
IPv6 Unicast Routing Prefix Structure, Historical	Table 25-4 386
IPv6 Well-Known Multicast Addresses, Important	Table 25-7 397
MIME application Media Type Subtypes	Table 76-8 1252
MIME audio Media Type Subtypes	Table 76-5 1251
MIME base64 Encoding Groups	Table 76-11 1259
MIME image Media Type Subtypes	Table 76-4 1250
MIME message Media Type Subtypes, Common	Table 76-10 1257
MIME model Media Type Subtypes	Table 76-7 1251
MIME multipart Media Type Subtypes, Common	Table 76-9 1253
MIME Standards	Table 76-2 1245
MIME text Media Type Subtypes	Table 76-3 1250
MIME video Media Type Subtypes	Table 76-6 1251
Mobile IP Mobility Agent Advertisement Extension Flags	Table 30-2 490
Mobile IP Mobility Agent Advertisement Extension Format	Table 30-1 489
Mobile IP Prefix-Lengths Extension Format	Table 30-3 490
Mobile IP Registration Reply Message Format	Table 30-6 495
Mobile IP Registration Request Message Format	Table 30-4 493
NAT, Bidirectional (Two-Way/Inbound), Operation of	Table 28-2 439
NAT (Overlapping NAT/Twice NAT), Operation of	Table 28-4 445
NAT, Port-Based (Overloaded), Operation of	Table 28-3 442
NAT, Unidirectional (Traditional/Outbound), Operation of	Table 28-1 435
NetBIOS Over TCP/IP Node Type (Option 46) Values	Table 63-11 1050
netstat Option Groups, Options, and Parameters (UNIX), Typical	Table 88-7 1480
netstat Option Groups, Options, and Parameters (Windows), Typical	Table 88-9 1482
netstat Universal Options and Parameters (UNIX), Typical	Table 88-8 1481
netstat Universal Options and Parameters (Windows), Typical	Table 88-10 1483
NFS External Data Representation (XDR) Data Types	Table 58-1 960
NFS Mount Protocol Server Procedures	Table 58-5 969
NFS Version 2 and Version 3 Server Procedures	Table 58-2 964
NFS Version 4 Server Operations	Table 58-4 966

	page
NFS Version 4 Server Procedures	Table 58-3 966
NNTP Base Commands	Table 85-5 1421
NNTP LIST Command Extensions	Table 85-7 1423
NNTP Newsreader Extensions	Table 85-8 1424
NNTP Reply Code Format: First Digit Interpretation	Table 85-9 1427
NNTP Reply Code Format: Second Digit Interpretation	Table 85-10 1428
NNTP Reply Codes	Table 85-11 1428
NNTP Transport Extensions	Table 85-6 1423
Non-ICMP Internet Standards That Define ICMP Messages	Table 31-1 509
nslookup Utility Commands, Typical	Table 88-5 1474
OSI Reference Model Layer Summary	Table 7-2 117
OSI Reference Model Real-World Analogy	Table 7-1 114
OSPF Common Header Format	Table 39-4 640
OSPF Database Description Message Flags	Table 39-7 642
OSPF Database Description Message Format	Table 39-6 642
OSPF Hello Message Format	Table 39-5 641
OSPF Link State Acknowledgment Message Format	Table 39-10 644
OSPF Link State Advertisement Header Format	Table 39-11 645
OSPF Link State Advertisement Header LS Types	Table 39-12 645
OSPF Link State Request Message Format	Table 39-8 643
OSPF Link State Update Message Format	Table 39-9 643
OSPF Link-State Database (LSDB), Sample	Table 39-1 629
OSPF LSDB with Costs, Sample	Table 39-2 634
OSPF Routes (Calculated), Example of	Table 39-3 637
ping (UNIX) Utility Options and Parameters, Common	Table 88-1 1466
ping (Windows) Utility Options and Parameters, Common	Table 88-2 1467
POP3 Transaction Commands	Table 78-1 1294
PPP Challenge Handshake Authentication Protocol (CHAP) Formats	Table 12-10 194
PPP Compression Control Protocol (CCP) Compression Algorithms	Table 11-1 171
PPP Control Message Format	Table 12-4 187
PPP Control Message Option Format	Table 12-6 189
PPP Control Messages, Code Values, and PPP Protocol Usage	Table 12-5 188
PPP Encryption Control Protocol (ECP) Compression Algorithms	Table 11-2 174
PPP Frames and Protocol Field Values, Common Protocols Carried in	Table 12-3 185
PPP General Frame Format	Table 12-1 182
PPP Link Control Protocol (LCP) Frame Types and Fields	Table 12-7 191
PPP Multilink Protocol Fragment Frame Format	Table 12-12 197
PPP PAP Authenticate-Request Frame Subfields	Table 12-9 194

	page
PPP Password Authentication Protocol (PAP) Frame Formats	Table 12-8 193
PPP Phases	Table 9-1 152
PPP Protocol Field Ranges	Table 12-2 184
PPP Standards	Table 9-2 153
Question Section QType Values	Table 57-6 935
Registration Request Flags	Table 30-5 494
Relative URL Specifications and Absolute Equivalents	Table 70-2 1152
RFC 822 Email Header Field Groups and Fields	Table 76-1 1239
RIP-1 Message Format	Table 38-1 615
RIP-1 RIP Entries	Table 38-2 615
RIP-2 Message Format	Table 38-3 619
RIP-2 Route Table Entries (RTEs)	Table 38-4 619
RIPng Message Format	Table 38-5 622
RIPng RTEs	Table 38-6 622
SMTP Commands	Table 77-2 1279
SMTP Extensions	Table 77-1 1276
SMTP Reply Code Format: First Digit Interpretation	Table 77-3 1281
SMTP Reply Code Format: Second Digit Interpretation	Table 77-4 1282
SMTP Reply Codes	Table 77-5 1283
SNMP Generic MIB Object Groups	Table 66-2 1095
SNMP MIB Modules, Common	Table 66-3 1096
SNMP PDU (Message) Classes	Table 67-1 1103
SNMP RMON MIB Object Groups	Table 69-1 1135
SNMP Security (SNMPsec) Standards	Table 65-3 1080
SNMP SMI Regular Data Types	Table 66-1 1089
SNMP Variable Binding Format	Table 68-1 1118
SNMPv1 Common PDU Format	Table 68-3 1120
SNMPv1 Error Status Field Values	Table 68-4 1120
SNMPv1 General Message Format	Table 68-2 1119
SNMPv1 Standards	Table 65-2 1080
SNMPv1 Trap-PDU Format	Table 68-5 1121
SNMPv2 Common PDU Format	Table 68-10 1126
SNMPv2 GetBulkRequest-PDU Format	Table 68-12 1129
SNMPv2 PDU Error Status Field Values	Table 68-11 1127
SNMPv2c General Message Format, Community-Based	Table 68-7 1124
SNMPv2c Standards, Community-Based	Table 65-4 1081
SNMPv2p General Message Format	Table 68-6 1123
SNMPv2p Standards, Party-Based	Table 65-1 1080

	page	
SNMPv2u General Message Format, User-Based	Table 68-8	1125
SNMPv2u Parameter Field Subfields	Table 68-9	1126
SNMPv2u Standards, User-Based	Table 65-5	1081
SNMPv3 General Message Format	Table 68-13	1130
SNMPv3 Msg Flags Subfields	Table 68-14	1131
SNMPv3 Scoped PDU Subfields	Table 68-15	1131
SNMPv3 Standards	Table 65-6	1082
Subnet ID of an IP Address, Determining Through Subnet Masking	Table 18-1	279
Subnet Masks for Class A, Class B, and Class C Networks, Default	Table 18-2	282
Subnetting Summary Table for Class A Networks	Table 18-3	289
Subnetting Summary Table for Class B Networks	Table 18-4	291
Subnetting Summary Table for Class C Networks	Table 18-5	292
 TCP Applications and Server Port Assignments, Common	 Table 46-1	 743
TCP Connection Termination Procedure	Table 47-4	764
TCP Finite State Machine (FSM) States, Events, and Transitions	Table 47-1	748
TCP Options	Table 48-4	773
TCP Pseudo Header for Checksum Calculations	Table 48-5	775
TCP Segment Control Bits	Table 48-2	772
TCP Segment Format	Table 48-1	771
TCP Segment Option Subfields	Table 48-3	773
TCP Simultaneous Connection Termination Procedure	Table 47-5	767
TCP Simultaneous Open Connection Establishment Procedure	Table 47-3	756
TCP Standards, Supplementary	Table 45-1	722
TCP Three-Way Handshake Connection Establishment Procedure	Table 47-2	754
TCP Transaction Example with Send and Receive Pointers	Table 48-6	785
TCP/IP Protocols	Table 8-1	132
TCP/IP Registered Port Numbers and Applications, Common	Table 43-2	709
TCP/IP Well-Known Port Numbers and Applications, Common	Table 43-1	708
Telnet NVT ASCII Control Codes (Standard), Interpretation of	Table 87-1	1445
Telnet Options, Common	Table 87-3	1451
Telnet Protocol Commands	Table 87-2	1448
TFTP Acknowledgment Message Format	Table 73-4	1213
TFTP Data Message Format	Table 73-3	1212
TFTP Error Message Format	Table 73-5	1214
TFTP OACK Message Format	Table 73-6	1214
TFTP Options	Table 73-1	1211
TFTP RRQ/WRQ Message Format	Table 73-2	1212
traceroute (UNIX) Utility Options and Parameters, Common	Table 88-3	1470
tracert (Windows) Utility Options and Parameters, Common	Table 88-4	1471

	page
Troubleshooting Protocols (TCP/IP), Miscellaneous	Table 88-15 1490
Truth Table, AND Operator	Table 4-11 74
Truth Table, Exclusive OR (XOR) Operator	Table 4-13 75
Truth Table, NOT Operator	Table 4-9 73
Truth Table, NOT Operator (Using Bit Values)	Table 4-10 74
Truth Table, OR Operator	Table 4-12 74
UDP and TCP, Summary Comparison of	Table 42-1 693
UDP Applications and Server Port Assignments, Common	Table 44-2 718
UDP Message Format	Table 44-1 715
URL Special Character Encodings	Table 70-1 1145
Usenet Big Eight Newsgroup Hierarchies	Table 85-1 1405
Usenet Headers, Common Additional	Table 85-4 1410
Usenet Mandatory Headers	Table 85-2 1409
Usenet Optional Headers	Table 85-3 1409



## **ACKNOWLEDGMENTS**

I dedicated this book to my wife and children to reflect the important role they have played in my life in general terms, and in accomplishing this book in particular. However, many others also contributed to the completion of this document, and I'd like to take a moment to acknowledge them.

I want to thank my “original” family: my father, Leon, and sisters, Cari and Cindy, for being supportive and lending a helpful ear about various issues during the time I’ve been engaged in this project. Thanks also to my “adoptive” family: Eli, Marge, Larry, and Steven. And I definitely want to thank the small group of close friends who have helped with ideas, advice, and much-needed laughs.

I would also like to specifically acknowledge the following individuals and organizations for their assistance:

- Bill Pollock, president and publisher of No Starch Press, for constantly expressing his faith in my abilities as an author, for being a sounding board, and for agreeing to publish this book. My thanks also to Susan Berge, Riley Hoffman, and everyone else at No Starch for putting up with me during this long project and helping make this book a reality.
- Adobe Systems Incorporated, for providing this relatively unknown author with two important pieces of software that I used in creating this book. First, Adobe FrameMaker, one of the best desktop publishing programs around, which was used to format and publish this document. Second, Adobe Photoshop, the industry-standard program for photo and graphics editing, which was used for processing graphics and other tasks.

- Frank Stearns, creator of the IXgen tool for FrameMaker. Without IXgen, it would have taken ten times longer to make the index for this book, and Frank himself was very generous with his time in answering questions from a newbie indexer (me!).
- SmartDraw.com, for the excellent SmartDraw diagramming software that was used to create most of the more than 300 illustrations that appear in this book.
- Fernando Gont and Barry Margolin, for their excellent technical review of *The TCP/IP Guide*, and their corrections and suggestions for improvement to the book.
- Tcat Houser, author and instructor, whose generosity, positive attitude, and enthusiasm for my writing helped boost my confidence as I worked to complete this project.
- All the regulars at The PC Guide Discussion Forums, for creating a fun community, keeping the site active, and agreeing to provide opinions on my writing. In fact, everyone who has supported The PC Guide and my other websites, financially and otherwise, helped make it possible for me to spend time on this project.

I've probably missed a few people who should be on this list; I hope all who are deserving of my appreciation will forgive their omission and accept my thanks.

## **A B O U T T H E A U T H O R**

I was born in 1966 in Windsor, Ontario, Canada and raised in nearby Toronto. I married my wife Robyn in 1990; we now live in southern Vermont with our three sons, Ryan (12), Matthew (9), and Evan (4).

I have had an interest in the field of computers ever since my early years, starting at the age of 14 when I received my first computer, an Apple ][, a gift from my parents. Since that time, I have worked in various computer-related fields in hardware and software. In 1989, I obtained a Bachelor of Applied Science from the University of Waterloo, in Waterloo, Ontario, Canada. I completed my formal education in 1993 with two master's degrees, in management and in electrical engineering and computer science (EECS), from MIT.

After a brief "conventional" technical career, I created and published The PC Guide, an extensive online reference work on personal computers, and in 1998, I decided to devote myself to my writing projects full time. The TCP/IP Guide was part of a larger networking project that I spent time on earlier this decade. I continue to work in the technical writing and editing field on various projects, for myself and other companies.

You may have noticed something missing here: no impressive listings of credentials. No, I'm not a *New York Times* best-selling author; I haven't been a professor at a prestigious Ivy League university for a quarter century; neither am I a top executive at a Silicon Valley giant. In some ways, I am a student of technology, just as you are. And my experience over the years has shown me that many of the people who know the most about how technology works have rather limited success in explaining what they know in a way that will

allow me to understand it. My interests, and I believe my skills, lie not in being an expert, but in serving as an *educator*, presenting complex information in a form that is sensible, digestible, and fun to read.

When I'm not working—all too rare these days—I spend time with my family and enjoy the peaceful quiet and natural beauty of the state of Vermont. I am also an avid amateur photographer, with interests particularly in nature and landscapes.

# INTRODUCTION

## Goals of *The TCP/IP Guide*

Every author who sets out to write a book or other document has certain objectives that he or she hopes to accomplish when the work is completed. This is why you can go into a library or bookstore, pick up several books that cover the same subject, and discover that they are surprisingly different—not just in their content or scope, but in their entire approach to the material.

I, too, had a number of goals when I set out to write this book. You certainly don't need to know them in order to read and appreciate the material, but understanding what I had in mind while I was writing may help you while you are reading. And if you are reading this information prior to buying *The TCP/IP Guide*, knowing what I strove for in writing the book may help you decide if this is the right resource for you.

My overall goal in writing this book was to create a resource that would allow anyone to obtain a deep understanding of how TCP/IP technologies really work. To accomplish this, I had a number of specific objectives that guided my writing efforts:

**Comprehensiveness** Like most authors writing a resource that covers a large subject, I wanted *The TCP/IP Guide* to be comprehensive. Of course, no single document can cover everything, so I have needed to limit the scope of the material. However, I feel I cover more about TCP/IP as a whole than any other single book or other resource.

**Comprehensibility** Creating a resource that is comprehensive is important, but I felt that it was even more important that the book be *comprehensible*. Over the past few years, I've had the opportunity to review many hundreds of books, guides, websites, and papers related to networking. I have found that even though most of them are generally high in quality, too many use unexplained technical jargon or assume extensive prior knowledge of networking concepts and technologies on the part of the reader. I worked very hard to ensure that my descriptions, even of very complex concepts, can be understood by almost every student of networking.

**Rationale** It's certainly important to know how every TCP/IP protocol functions. However, to gain a true understanding of complex material, one also needs to understand the reasons behind why things are what they are. In writing this material, I have always tried to explain not just the *what*, but also the *why* of TCP/IP. I have anticipated and answered questions that I believe might commonly arise in the mind of someone learning about this technology.

**Illustrations** A picture is worth a thousand words, as they say. There are many concepts that no amount of verbiage will adequately explain, while a simple illustration will do the trick. For this reason, I spent many months creating more than 300 diagrams (some simple and some not so simple!) to complement the written material in *The TCP/IP Guide*.

**User-friendliness** I have intentionally broken many of the rules of conventional book authorship, in creating a document that uses a conversational, first-person style, and no small amount of humor where appropriate. My intention was to make you feel at home while you read material that can be quite technically difficult. I want you to think of me as a friend sitting next to you at your computer explaining how TCP/IP works, rather than a professor preaching at you from a podium.

**Organization** Many networking books consist of dozens of subjects just listed one after the other, leaving the reader to wonder how everything fits together. When I first began this book, I spent weeks just organizing it, with the result being a structure that indicates clearly how subjects are interrelated. I also carefully laid out each individual section to ensure that it covered its topic in a way that made sense.

**Multiple levels of detail** I realize that some people reading a TCP/IP book might want only a quick summary of the operation of its constituent protocols, while others want to learn all the nuances of how everything works. I have provided the full details that most readers will want, while also including overview topics in each chapter that summarize each technology for quick perusal. This gives you the option of either skimming the surface or "diving deep," as you choose.

**Platform independence** I have endeavored whenever possible to avoid describing TCP/IP in terms specific to any hardware or software platform. Even though I use a PC for most of my computing and UNIX for some tasks, most of the material is not particular to any type of device or operating system (though I do focus more on networks of smaller computers than larger ones).

How successful was I in achieving these goals? I'd like to think I did a pretty good job, but ultimately, you will be the judge!

## Scope of The TCP/IP Guide

The first step to dealing with a problem is recognizing that you have one. So, I have to come clean with you, my reader. I have a problem: an addiction to . . . detail. Every time I set out to write about a particular protocol, technology, or concept, I start with a modest goal regarding how much I want to write. I always begin knowing that I really need to control myself, to prevent my project from going on forever. But as I explore each subject, I learn more and more, and I start to say to myself things like, “This is important. I simply *must* include coverage for it,” and, “If I’m going to cover subject #1, I also should cover subject #2, because they are related.” This is how I turn six-month projects into multiyear ordeals.

However, even though self-control in this area is a weakness for me, even *I* realized I could not possibly cover *everything* related to TCP/IP in this book. Consider that the TCP/IP suite contains dozens of protocols and technologies, each written about in thick books. I was willing to spend years on this project, but not decades! Thus, I had to limit the scope of this book somewhat, both to preserve what remains of my sanity and to spare you from having to wade through a ridiculously large document.

Here are a few different points that will help explain decisions that I made to limit the scope of *The TCP/IP Guide*:

**Theory versus practice** This is primarily a *reference resource* on the TCP/IP protocol suite. The material here is designed to allow a student to learn the nuts and bolts of how TCP/IP works. I do discuss quite a number of “real-world” practical issues related to how TCP/IP internetworks operate, but this is not my primary focus here. If you want to really understand what TCP/IP is and what makes it work, you’ve come to the right place. If all you want is simple instructions on how to connect a few PCs together in your home using TCP/IP, this probably isn’t the book for you.

**Current versus future protocols** Most of the emphasis in this book is on the present state of the art in TCP/IP. The suite is always changing, new protocols are constantly being written, and revisions to existing protocols continue to be published. I have not provided extensive coverage of technologies still in development, to try to keep the size of the book manageable and to prevent the book from being out-of-date before it even hits the store shelves. The one exception to this general rule of thumb is version 6 of the Internet Protocol (IPv6), which represents a significant change to the core of how most of TCP/IP operates. While not universally deployed yet, IPv6 is sufficiently far along in its development that I feel any student of TCP/IP needs to know what it is, learn how it works, and understand its significance. Thus, I have included several detailed chapters on IPv6, and also mentioned how it impacts the operation of several other key protocols such as the Internet Control Message Protocol (ICMP), Domain Name System (DNS), and Dynamic Host Configuration Protocol (DHCP).

**Application coverage** Many thousands of different applications run on TCP/IP internetworks, and I could not possibly hope to describe all of them. The scope of this book is limited to the most important, “classic” TCP/IP applications and application protocols, such as electronic mail, general file transfer, and the World Wide Web.

**TCP/IP versus the Internet** The TCP/IP protocol suite and the Internet are very closely related in many ways, as you will discover as you read this book. In fact, they are often tied together so much that it is hard to discuss one without the other. However, the Internet as a whole is an enormous subject, and trying to describe it in general terms would have substantially increased the size of this book. Thus, I describe Internet issues only within the context of explanations of TCP/IP technologies. For example, while I cover the World Wide Web in this book, I discuss its generalities only briefly. I focus my technical discussions on how the Hypertext Transfer Protocol (HTTP) that implements it works. I don’t talk about how to set up a website, how to choose a web browser, or any of those sorts of details. Those subjects are covered in a dazzling array of different books, papers, and, of course, websites.

**Limited TCP/IP security coverage** Security is a very important and large topic, especially in modern networking. This book does include a fairly detailed section on the operation of the IP Security protocol (IPSec), and also touches on security issues in describing several other protocols and technologies. However, it is not specifically geared toward detailed discussions of security considerations.

**Small computer orientation** In general terms, TCP/IP technologies can be used to connect together any types of devices that have the appropriate hardware and software. There are some issues, however, where explanations require me to focus on how specific types of underlying networks and devices work; this is especially true of some of my diagrams. In these cases, my preference has generally been to show how TCP/IP is used to connect together typical small computers such as PCs, Macintoshes, and UNIX workstations, which are what most people use.

## The *TCP/IP Guide* Features

I created *The TCP/IP Guide* to provide you with an unparalleled breadth and depth of information about TCP/IP. This meant including a lot of content in this book—it has 88 chapters, several hundred sections, and more than 1,600 pages. However, I recognized as I was writing this tome that the real goal is not just to provide a lot of detailed information, but also to present it in such a way that it can be easily understood by you, the reader. This requires more than just writing large amounts of text and putting it all into a big file.

For this reason, I have incorporated a number of special features into this book to help make it easier for you to “digest.” These include the special structure of the book, special inserts to help you remember topics, and more.

First, *The TCP/IP Guide* uses a three-level structure to organize its content. The book as a whole is divided into three overall *sections*, covering overview/background information, lower-layer protocols, and higher-layer application protocols,

respectively. Within each section is a number of *parts*, which group together related chapters. Each chapter is, in turn, structured with sections and subsections to present the material in the most understandable way possible.

*The TCP/IP Guide* contains more than 300 detailed illustrations, which support the textual descriptions of TCP/IP technologies and protocols, and help make sense of difficult concepts. Most include brief descriptions that allow you to quickly understand what the illustration means without even needing to read the full surrounding text. The book also has more than 300 tables, which present large amounts of information in an organized and readable manner or highlight examples.

Most of the discussions in *The TCP/IP Guide* are presented as free-form text, as you would expect in any document. However, I use *notes* when I need to clarify or explain something that I feel you need to know, but which is either not directly related to the topic under discussion or is sufficiently “tangential” that it needs to be separated from the main text to avoid disrupting its flow. Examples include terminology explanations, “sidebar” historical discussions, anecdotes, and clarifications relating to how I am describing particular concepts.

Last, but definitely not least, are the *key concepts* inserts. I tried very hard in crafting this book to provide a variety of ways to present information, to better suit the learning styles of different readers. To that end, I have created hundreds of these special paragraphs, which summarize and highlight the most important concepts and essential pieces of knowledge in *The TCP/IP Guide*. They can be very useful for quickly distilling the essence of a topic without reading an entire explanation, or for refreshing your memory of a subject. Obviously, however, they contain few details, so you should not assume that you fully understand a topic or concept just by reading this sort of summary.

## The TCP/IP Guide Online!

This book had its origins as an online website, called appropriately enough, The TCP/IP Guide. The site is still active and presents the same information as this book contains, albeit in a slightly different format and structure.

You may find the online version of the site useful if you are traveling or need to look up TCP/IP information quickly when you don't have the book handy. As a further bonus, the pages of the site are extensively hyperlinked for ease of use, and the diagrams are in full color. The online version also has a full hyperlinked table of contents and a search engine! Visit the site at:

<http://www.TCPIPGuide.com>.

## Your Feedback and Suggestions

One of the ways that this book differs from the typical technical reference book is that it is a very *personal* work. When you read the material here, I want you to feel as though I am explaining the many technologies and concepts to you personally, because that's how I feel when I am writing. A published book of this sort is by its

nature a type of “one-way” communication, from me to you; however, I am also interested in what you have to say to me. For this reason I strongly encourage you to provide me with feedback on this book and suggestions that you may have for it.

First, let me point out that all books have mistakes, and despite undergoing a rigorous technical review and multiple-stage editing process, this book is probably no exception. Before contacting me regarding an error or problem with this book, please check the errata page to see if what you found has already been reported. You can find that page here: <http://www.tcpipguide.com/bookerrata.htm>. I welcome your constructive criticisms and suggestions. If there is something you don’t like about the material, please tell me. Even better, make a suggestion for how to improve it. I am also happy to read your compliments or complaints, and I will gladly answer questions that pertain directly to the use of the book. You can contact me at [tcpipbook@tcpipguide.com](mailto:tcpipbook@tcpipguide.com).

# **SECTION I**

## **TCP/IP OVERVIEW AND BACKGROUND INFORMATION**

They say the best place to start is at the beginning, and that's exactly where you are now. This initial section contains background information that will help you to understand what networking is about and where TCP/IP fits into the grand scheme of things. This introductory information will help ease you into your studies of TCP/IP, and it is particularly valuable to those who are new to the world of networking.

This section contains three parts. The first part covers a number of important fundamental aspects of networks, discussing how they are used, the standards that define them, the terminology that describes them, and much more. The second part describes the important OSI Reference Model, which is an essential tool to comprehending the function and organization of networking technologies. The third part contains a high-level overview of the TCP/IP protocol suite, which will frame the more complete discussions of individual TCP/IP protocols that follow in the latter two sections of this book.

Let's get started!



# PART I-1

## **NETWORKING FUNDAMENTALS**

Unlike authors of many other TCP/IP-related resources, I do not assume that readers already know what networking is all about. After all, that's why you are reading this book!

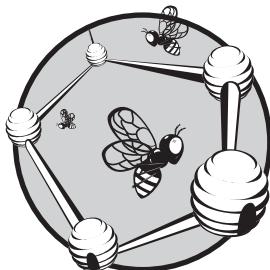
This part provides an overview of some of the basic issues related to networking. It includes discussions of some of the most fundamental networking concepts and ideas. It serves not only to provide you with useful background material, but also as a repository for general information, so that I don't need to repeat it in many different places elsewhere in the book (and if you already know about these basics, you don't need to skip over them in many other locations). The topics covered here are useful for understanding certain TCP/IP concepts. However, some of the material is very broadly oriented toward networking as a whole and is not specific to TCP/IP internetworking.

This part consists of four chapters. The first chapter in this part introduces networking in broad terms, describes its fundamental characteristics, and differentiates between network sizes and types. The second chapter talks about many different matters related to network performance. The third chapter explains the importance of networking standards and standards organizations. Finally, the fourth chapter provides background information about how data is stored and manipulated in computers; if you are new to computing, you may find this information useful when reading some other parts of this book. If you are experienced in networking and related

technologies, you may wish to skip this part of the book. Or, you can scan the headings in the chapters; if you understand the terminology mentioned in a heading, you can probably skip the discussion. Cross-references in other areas of the book refer to information in this part as appropriate, so if you need to fill in your knowledge of a particular fundamental on the fly, you can do so rather easily.

# 1

## **NETWORKING INTRODUCTION, CHARACTERISTICS, AND TYPES**



Someone new to networking will usually have some pretty important questions.

What is networking all about? What are the most important attributes that describe networks? And what sort of networks exist? The obvious place to begin discussing networking fundamentals is to answer those questions using a high-level introduction to networking as a whole.

This chapter is divided into three sections. The first provides a quick introduction to networking. I define networking in the most basic terms, then place networking in an overall context by describing some of its advantages and benefits, as well as some of its disadvantages and costs.

The second section discusses key concepts that describe and differentiate between types of networks and networking technologies. This is where I'll define terms and "buzzwords" that you cannot avoid if you are going to learn about networks. The topics here include explanations of

protocols, switching methods, types of network messages, message formatting, and ways of addressing messages. I also discuss the differences between client-server and peer-to-peer networking.

In the final section, I describe the major types of networks by drawing distinctions between them based on their size and scope, and I also show you how to use each type and size. I discuss LANs, WLANs, and WANs, and a few variations on these three main categories. I also explore the many terms that are related to the various sizes of networks and how they are used, including segments, subnetworks, internetworks, intranets, and extranets.

## Introduction to Networking

In this day and age, networks are everywhere, especially in the form of the Internet. The Internet, the ultimate network, has revolutionized not only the computer world, but the lives of millions. We tend to take for granted that computers should be connected together. In fact, these days, whenever I have two computers in the same room, I have a difficult time *not* connecting them!

Given the ubiquitousness of networking, it's hard to believe that the field is still a relatively young one, especially when it comes to hooking up PCs. In approaching any discussion of networking, it is very useful to take a step back and look at networking from a higher level. What is it, exactly, and why is it now considered so important that it is just assumed that most PCs and other devices will be networked?

### What Is Networking?

For such an extensive and involved subject that includes so many different technologies, hardware devices, and protocols, networking is actually quite simple. A *network* is simply a collection of computers or other hardware devices that are connected together, either physically or logically, using special hardware and software that allows the devices to exchange information and cooperate. *Networking* is the term that describes the processes involved in designing, implementing, upgrading, managing, and otherwise working with networks and network technologies.

**KEY CONCEPT** A network is a set of hardware devices connected together, either physically or logically. This allows them to exchange information.

Networks are used for an incredible array of purposes. Most people learning about networking think about networking as interconnecting PCs and other “true” computers, but you use a variety of types of networks every day. Each time you pick up a phone, use a credit card at a store, get cash from an ATM machine, or even plug in an electrical appliance, you are using some type of network.

In fact, the definition can even be expanded beyond the world of technology. I'm sure you've heard the term *networking* used to describe the process of finding an employer or employee through friends and associates. Similarly, the idea here is that independent units are connected together to share information and cooperate.

The widespread networking of personal computers is a relatively new phenomenon. For the first decade or so of their existence, PCs were very much “islands

unto themselves,” and were rarely connected together. In the early 1990s, PC networking began to grow in popularity as businesses realized the advantages that networking could provide. By the late 1990s, networking in homes with two or more PCs really started to take off as well.

This interconnection of small devices represents, in a way, a return to the good old days of mainframe computers. Before computers were small and personal, they were large and centralized machines that many users operating remote terminals shared. Although having all of that computer power in one place had many disadvantages, one benefit was that all users were connected because they shared the central computer.

Individualized PCs took away that advantage. Networking attempts to move computing to a middle ground. It provides PC users with the best of both worlds: the independence and flexibility of personal computers, and the connectivity and resource sharing of mainframes. In fact, networking today is considered so vital that it’s hard to conceive of an organization with two or more computers that would not want to connect them together!

### ***The Advantages and Benefits of Networking***

You have undoubtedly heard the expression “The whole is greater than the sum of its parts.” This phrase describes networking very well and explains why it has become so popular. A network isn’t just a bunch of computers with wires running between them. Properly implemented, a network is a system that provides its users with unique capabilities, above and beyond what the individual machines and their software applications can provide.

Most of the benefits of networking can be divided into two basic categories: *connectivity* and *sharing*. Networks allow computers, and hence their users, to connect to each other. They also allow for the easy sharing of information and resources, and for the simple cooperation between the devices in other ways. Since modern business depends so much on the intelligent flow and management of information, this ease of use tells you a lot about why networking is so valuable.

Here, in no particular order, are some of the specific advantages generally associated with networking:

**Connectivity and Communication** Networks connect computers and the users of those computers. Individuals within a building or workgroup can be connected through *local area networks (LANs)*; LANs in distant locations can be interconnected to form larger, *wide area networks (WANs)*. Once computers are connected, it is possible for network users to communicate with each other using technologies such as electronic mail. This makes the transmission of business (or nonbusiness) information easier, more efficient, and less expensive than it would be without the network.

**Data Sharing** One of the most important uses of networking is to allow the sharing of data. Before networking was common, an accounting employee who wanted to prepare a report for her manager would have to produce it on her PC, put it on a floppy disk, and then walk it over to the manager, who would transfer the data to her PC’s hard disk. (This sort of “shoe-based network” was sometimes sarcastically called a *sneakernet*.)

True networking allows thousands of employees to share data much more easily and quickly than this. It also makes possible applications that enable many people to access and share the same data, such as databases, group software development, and much more.

**Hardware Sharing** Networks facilitate the sharing of hardware devices. For example, instead of giving each employee in a department an expensive color printer (or resorting to the sneakernet again), you can place one printer on the network for everyone to share.

**Internet Access** The Internet is itself an enormous network, so whenever you access the Internet, you are using a network. The significance of the Internet today is hard to exaggerate!

**Internet Access Sharing** Small computer networks allow multiple users to share a single Internet connection. Special hardware devices allow the bandwidth of the connection to be easily allocated to various individuals as they need it, and these devices permit an organization to purchase one high-speed connection instead of many slower ones.

**Data Security and Management** In a business environment, a network allows the administrators to manage the company's critical data better. Instead of spreading data over dozens or even hundreds of small computers in a haphazard fashion as users create it, administrators can centralize data on shared servers. This makes it easy for everyone to find the data and makes it possible for the administrators to ensure that the data is regularly backed up. Administrators can also implement security measures to control who can read or change various pieces of critical information.

**Performance Enhancement and Balancing** Under some circumstances, you can use a network to enhance the overall performance of some applications by distributing the computation tasks to various computers on the network.

**Entertainment** Networks facilitate many types of games and entertainment. The Internet itself offers many sources of entertainment. In addition, many multiplayer games operate over a LAN. Many home networks are set up for this reason, and gaming across WANs (including the Internet) has also become quite popular. Of course, if you are running a business and have employees who are easily amused, you might insist that this is really a *disadvantage* of networking rather than an advantage!

**KEY CONCEPT** At a high level, networks are advantageous because they allow computers and people to be connected together so that they can share resources. Some of the specific benefits of networking include communication, data sharing, Internet access, data security and management, application performance enhancement, and entertainment.

## ***The Disadvantages and Costs of Networking***

Now that I have discussed the great value and many useful benefits of networking, I must bring you crashing back to Earth with that old nemesis of the realistic: TANSTAAFL. For those who are not Heinlein fans, this acronym stands for "There

ain't no such thing as a free lunch.” Even though networking really does represent a whole that is greater than the sum of its parts, it does have some real and significant costs and drawbacks associated with it.

Here are a few disadvantages of networking:

**Network Hardware, Software, and Setup Costs** Computers don’t just magically network themselves, of course. Setting up a network requires an investment in hardware and software, as well as funds for planning, designing, and implementing the network. For a home with a small network of two or three PCs, this is relatively inexpensive. It amounts to more or less a hundred dollars with today’s low prices for network hardware, and practically no setup costs considering that the operating systems have already been designed for networks. For a large company, however, costs can easily run into tens of thousands of dollars or more.

**Hardware and Software Management and Administration Costs** In all but the smallest of implementations, ongoing maintenance and management of the network requires the care and attention of an IT professional. In a smaller organization that already has a system administrator, a network may fall within this person’s job responsibilities, but it will take time away from other tasks. In more substantial organizations, a network administrator may need to be hired, and in large companies an entire department may be necessary.

**Undesirable Sharing** With the good comes the bad; though networking allows the easy sharing of useful information, it also allows the sharing of undesirable data. One significant sharing problem in this regard has to do with viruses, which are easily spread over networks and the Internet. Mitigating these effects costs time, money, and administrative effort.

**Illegal or Undesirable Behavior** Similar to the previous point, networking facilitates useful connectivity and communication, but also brings difficulties with it. Typical problems include the abuse of company resources, distractions that reduce productivity, the downloading of illegal or illicit materials, and even software piracy. In larger organizations, these issues must be managed through explicit policies and monitoring, which, again, further increases management costs.

**Data Security Concerns** If a network is implemented properly, it is possible to greatly improve the security of important data. In contrast, a poorly secured network puts critical data at risk, exposing it to the potential problems associated with hackers, unauthorized access, and even sabotage.

Most of these costs and potential problems can be managed by those who set up and run networks. In the end, the choice of whether to use a network is a matter of weighing the advantages against the disadvantages. Today, nearly everyone decides that networking *is* worthwhile.

**KEY CONCEPT** Networking has a few drawbacks that you can weigh against its many positive aspects. Setting up a network has costs in hardware, software, maintenance, and administration. It is also necessary to manage a network to keep it running smoothly and to address possible misuse or abuse issues. Data security also becomes a much bigger concern when computers are connected together.

## Fundamental Network Characteristics

There are many different kinds of networks and network technologies that are used to create them. The proliferation of networking methods has generally occurred for a very good reason: Different needs require different solutions. The drawback of this is that there are so many different types of protocols and technologies for the networking student to understand!

Before you can really compare these approaches, you need to understand some of the basic characteristics that make networks what they are. Although network types may be quite dissimilar, they are often described and even contrasted on the basis of a number of common attributes, which I'll discuss in the following sections.

### ***Networking Layers, Models, and Architectures***

One of the reasons why many people find networking difficult to learn is that it can be a very complicated subject. One of the chief reasons for this complexity is that networks consist of so many hardware and software elements. While a network user may perceive that he is using only one computer program (like a web browser) and one piece of hardware (like a PC), these are parts of a much larger puzzle. In order for even the simplest task to be accomplished on a network, dozens of different components must cooperate by passing control information and data to accomplish the overall goal of network communication.

The best way to understand any complex system is to break it down into pieces and then analyze what those pieces do and how they interact. The most logical approach is to divide the overall set of functions into modular components, each of which is responsible for a particular function. At the same time, you also need to define interfaces between these components, which describe how they fit together. This enables you to simplify the complexity of networking by approaching it in digestible chunks.

Networking technologies are most often compartmentalized in this manner by dividing their functions into *layers*, each of which contains hardware and software elements. Each layer is responsible for performing a particular type of task and interacts with the layers above and below it. Layers are conceptually arranged into a vertical *stack*. Lower layers are charged with more concrete tasks such as hardware signaling and low-level communication; they provide services to the higher layers. The higher layers, in turn, use these services to implement more abstract functions such as implementing user applications.

Dividing networks into layers this way is somewhat like the division of labor in a manufacturing facility, and it yields similar benefits. Each hardware device or software program can be specialized to perform the function needed by that layer, like a well-trained specialist on an assembly line. The different modules can be combined in different ways as needed. This way, it's also easier to understand how a network functions overall.

One other important benefit of layering is that makes it possible for technologies defined by different groups to interoperate. For this to be possible, it is necessary for everyone to agree on how layers will be defined and used. The most common tool for this purpose is a *networking model*. The model describes what the

different layers are in the network, what each is responsible for doing, and how they interact. A universally accepted model ensures that everyone is on the same page when creating hardware and software.

The most common general model in use today is the Open Systems Interconnection (OSI) Reference Model, which consists of seven stacked layers. These range from the physical layer (layer 1) at the bottom, which is responsible for low-level signaling, to the application layer (layer 7) at the top, where application software is implemented. Understanding the OSI model is essential to understanding networking as a whole. I explain models and layers in more detail, and provide a complete description of the OSI Reference Model, in Part I-2 of this book.

Closely related to the concept of a model is the concept of an *architecture*. An architecture is essentially a set of rules that describes the function of some portion of the hardware and software that constitutes a stack of layers. Such a ruleset usually takes the form of a specification or standard that describes how equipment and programs using the technology must behave. A networking architecture is designed to implement the functions associated with a particular contiguous set of layers of the OSI Reference Model, either formally or informally.

In this book, I discuss TCP/IP, the protocol suite that runs the Internet. TCP/IP is a complex set of technologies that spans many layers of the OSI model. By examining the various components of TCP/IP and how they implement different OSI model layers, you will really learn how TCP/IP works. For starters, the name of the suite, TCP/IP, comes from the Transmission Control Protocol (TCP), which operates at layer 4 of the OSI model, and the Internet Protocol (IP), which runs at OSI model layer 3. IP provides services to layer 4 and uses services from layer 2 below it. TCP uses IP's functions and provides functions to the layers above it.

I'll start a more complete examination of TCP/IP by looking at its architecture, and by looking at a second, special model that was developed specifically to make sense of TCP/IP. Both are explored in Chapter 8.

## **Protocols: What Are They, Anyway?**

If there's one word you will get used to seeing a lot as you go through this book, it is *protocol*. You will see references to networking protocols, internetworking protocols, high-level protocols, low-level protocols, protocol stacks, protocol suites, subprotocols, and so on. Clearly, protocols are important, yet many reference works and standards use the term over and over again without ever explaining it. One reason for this may be because the term is somewhat vague and can have many meanings.

In some cases, understanding a technical term is easier if you go back to look at how the term is used in plain English. In the real world, a protocol often refers to a code of conduct or a form of etiquette observed by diplomats. These people must follow certain rules of ceremony and formality to ensure that they communicate effectively without causing conflict. They also must understand what is expected of them when they interact with representatives from other nations, making sure that, for example, they do not offend anyone due to an unfamiliarity with local customs. In fact, most people follow various protocols; they are sort of the unwritten rules of society.

This may seem to have little to do with networking, but in fact, this is a pretty good high-level description of what networking protocols are about. They define a language and a set of rules and procedures that enable devices and systems to communicate. Obviously, computers do not have local customs, and they hardly have to worry about committing a faux pas that might cause another computer to take offense. Networking protocols concern themselves with ensuring that all the devices on a network or internetwork are in agreement about how various actions must be performed in the total communication process.

A protocol is thus basically a way of ensuring that devices are able to talk to each other effectively. In most cases, an individual protocol describes how communication is accomplished between one particular software or hardware element in two or more devices.

In the context of the OSI Reference Model, a protocol is formally defined as a set of rules governing communication between entities at the same layer. For example, TCP is responsible for a specific set of functions on TCP/IP networks. Each host on a TCP/IP network has a TCP implementation, and those hosts all communicate with each other logically at layer 4 of the OSI model.

**NOTE** *The formalized OSI Reference Model meaning of the word protocol is covered in the OSI Reference Model topic on horizontal layer communication (discussed in Chapter 5, in the section titled “Protocols: Horizontal (Corresponding Layer) Communication”).*

While OSI Reference Model definitions are sometimes overly theoretical in nature, this definition of protocol is rather accurate in assessing protocols in real-world networking. If something doesn’t specify a means of communication, it arguably isn’t a protocol.

**KEY CONCEPT** A networking protocol defines a set of rules, algorithms, messages, and other mechanisms that enables software and hardware in networked devices to communicate effectively. A protocol usually describes a means for communication between corresponding entities at the same OSI Reference Model layer in two or more devices.

Despite this, the term *protocol* is often used colloquially to refer to many different concepts in networking. Some of the more common alternative uses of the word are listed here:

**Protocol Suites** It is very common to hear the word *protocol* used to refer to sets of protocols that are more properly called *protocol suites* (or *stacks*, in reference to a stack of layers). For example, TCP/IP is often called just a protocol when it is really a (large) set of protocols.

**Microsoft Windows Protocols** One important example of the issue of referring to protocol suites as single protocols is the networking software in Microsoft Windows. It usually calls a full networking stack like TCP/IP or IPX/SPX just a protocol. When you install one of these so-called protocols, however, you actually get a software module that supports a full protocol suite.

**Other Technologies** Sometimes technologies that are not protocols at all are called protocols, either out of convention or perhaps because people think it sounds good. For example, TCP/IP Remote Network Monitoring (RMON) is often called a protocol when it is really just an enhancement to the Simple Network Management Protocol (SNMP), which *is* a protocol! (See Part III-4 for details on SNMP and RMON.)

So, does it really matter whether a protocol is a “true” protocol or not? Well, the networking hardware devices and software programs sure don’t care. But hopefully, having read about the term and what it means, you will be able to better understand the word when you encounter it in your studies—especially in the places where it may not always be used in a way that’s entirely consistent with its formal definition.

## **Circuit-Switching and Packet-Switching Networks**

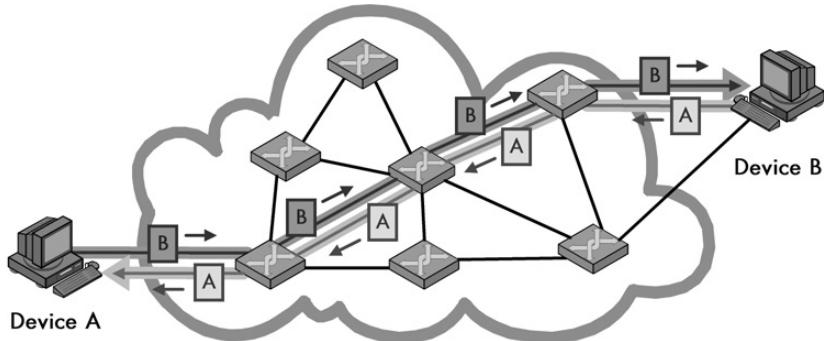
Networks are devices that are connected together using special hardware and software that allows them to exchange information. The most important word in that sentence is the final one: *information*. As you will see in your exploration of this book, there are many methods for exchanging information between networked devices. There are also a number of ways of categorizing and describing these methods and the types of networks that use them.

One fundamental way to differentiate between networking technologies is on the basis of the method used to determine the path between devices over which information will flow. In highly simplified terms, there are two approaches: a path can be set up between the devices in advance, or the data can be sent as individual data elements over a variable path.

### **Circuit Switching**

In the *circuit-switching* networking method, a connection called a *circuit*, which is used for the whole communication, is set up between two devices. Information about the nature of the circuit is maintained by the network. The circuit may be either a fixed one that is always present or one that is created on an as-needed basis. Even if many potential paths through intermediate devices may exist between the two devices that are communicating, only one will be used for any given dialogue, as shown in Figure 1-1.

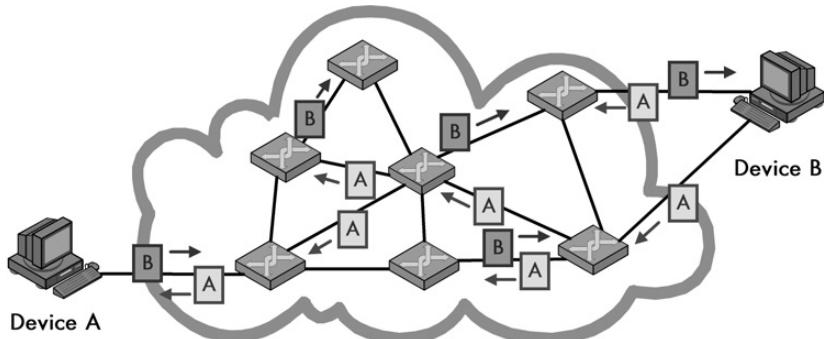
The classic example of a circuit-switched network is the telephone system. When you call someone and she answers, you establish a circuit connection and can pass data in a steady stream. That circuit functions the same way, regardless of how many intermediate devices are used to carry your voice. You use it for as long as you need it and then terminate the circuit. The next time you call, you get a new circuit, which may (probably will) use different hardware than the first circuit did, depending on what’s available at that time in the network.



**Figure 1-1: Circuit switching** In a circuit-switched network, before communication can occur between two devices, a circuit is established between them. This is shown as a darker line for the conduit of data from Device A to Device B, and a matching lighter line from B back to A. Once it's set up, all communication between these devices takes place over this circuit, even though there are other possible ways that data could conceivably be passed over the network of devices between them. Contrast this diagram to Figure 1-2.

### Packet Switching

In the *packet-switching* network type, no specific path is used for data transfer. Instead, the data is chopped up into small pieces called *packets* and sent over the network. You can route, combine, or fragment the packets as required to get them to their eventual destination. On the receiving end, the process is reversed—the data is read from the packets and reassembled to form the original data. A packet-switched network is more analogous to the postal system than it is to the telephone system (though the comparison isn't perfect). An example is shown in Figure 1-2.



**Figure 1-2: Packet switching** In a packet-switched network, no circuit is set up prior to sending data between devices. Blocks of data, even from the same file or communication, may take any number of paths as they journey from one device to another. Compare this to Figure 1-1.

**KEY CONCEPT** One way that networking technologies are categorized is based on the path used to carry data between devices. In circuit switching, a circuit is first established and then used to carry all data between devices. In packet switching, no fixed path is created between devices that communicate; it is broken into packets, each of which may take a separate path from sender to recipient.

## Which Switching Method to Choose?

A common temptation when considering alternatives such as these is to ask which is better; the answer is neither. There are places for which one is more suited than the other, but if one were clearly superior, both methods wouldn't be used.

One important issue in selecting a switching method is whether the network medium is *shared* or *dedicated*. Your phone line can be used for establishing a circuit because you are the only one who can use it—assuming you can keep that pesky wife/husband/child/sister/brother/father/mother off it. However, this doesn't work well with LANs, which typically use a single shared medium and baseband signaling. If two devices were to establish a connection, they would lock out all the other devices for a long period of time. It makes more sense to chop the data into small pieces and send them one at a time. Then, if two other devices want to communicate, *their* packets can be interspersed, and everyone can share the network.

The ability to have many devices communicate simultaneously without dedicated data paths is one reason why packet switching is becoming predominant today. However, there are some disadvantages of packet switching compared to circuit switching. One is that since all data does not take the same predictable path between devices, it is possible that some pieces of data may get lost in transit or show up in the incorrect order. In some situations this does not matter, but in others it is very important indeed.

Although the theoretical difference between circuit and packet switching is pretty clear-cut, understanding how to use them is a bit more complicated. One of the major issues is that in modern networks, they are often combined.

For example, suppose you connect to the Internet using a dial-up modem. You will be using IP datagrams (packets) to carry higher-layer data, but it will be over the circuit-switched telephone network. Yet the data may be sent over the telephone system in digital packetized form. So in some ways, both circuit switching and packet switching are being used concurrently.

Another issue is the relationship between circuit and packet switching, and whether a technology is connection-oriented or connectionless. The two concepts are related but not the same, as you will see in a moment.

**NOTE** *The word packet is only one of several terms that are used to refer to messages that are sent over a network. Other terms that you will encounter include frame, datagram, cell, and segment. You will learn more about these terms later in this chapter.*

## Connection-Oriented and Connectionless Protocols

I just compared networking technologies based on whether or not they use a dedicated path or *circuit* over which they send data. Another way in which technologies and protocols are differentiated has to do with whether or not they use *connections* between devices. This issue is closely related to the matter of packet versus circuit switching.

Protocols are divided into the following two categories based on their use of connections:

**Connection-Oriented Protocols** These protocols require you to establish a logical connection between two devices before transferring data. This is generally accomplished by following a specific set of rules that specify how a connection should be initiated, negotiated, managed, and eventually terminated. Usually, one device begins by sending a request to open a connection, and the other responds. The devices pass control information to determine if and how the connection should be set up. If this is successful, data is sent between the devices. When they are finished, the connection is broken.

**Connectionless Protocols** These protocols do not establish a connection between devices. As soon as a device has data to send to another, it just sends it.

**KEY CONCEPT** A connection-oriented protocol is one in which a logical connection is first established between devices prior to data being sent. In a connectionless protocol, data is just sent without a connection being created.

You can probably immediately see the relationship between the concepts of circuits and connections. Obviously, in order to establish a circuit between two devices, you must connect them. For this reason, circuit-switched networks are inherently based on connections. This has led to the interchangeable use of the terms *circuit-switched* and *connection-oriented*.

However, this is an oversimplification that results from a common logical fallacy—people make the mistake of thinking that if A implies B, then B implies A, which is like saying that since all apples are fruit, then all fruit are apples! A connection is needed for a circuit, but a circuit is *not* a prerequisite for a connection. There are, therefore, protocols that are connection-oriented, even though they aren't predicated on the use of circuit-based networks at all.

These connection-oriented protocols are important because they enable the implementation of applications that require connections over packet-switched networks that have no inherent sense of a connection. For example, to use the TCP/IP File Transfer Protocol (FTP), you want to be able to connect to a server, enter a login and password, and then execute commands to change directories, send or retrieve files, and so on. This requires the establishment of a connection over which commands, replies, and data can be passed. Similarly, the Telnet Protocol (TP) involves establishing a connection—it lets you remotely use another machine. Yet, both of these work (indirectly) over IP, which is based on the use of packets, through the important principle of layering (see Chapter 5).

**KEY CONCEPT** Circuit-switched networking technologies are inherently connection-oriented, but not all connection-oriented technologies use circuit switching. Logical connection-oriented protocols can be implemented on top of packet-switching networks to provide higher-layer services to applications that require connections.

To comprehend the relationship between connections and circuits, you must recall the layered nature of modern networking architecture (as I discuss in some detail in Chapter 5). Even though packets may be used at lower layers for the mechanics of sending data, a higher-layer protocol can create logical connections through the use of messages sent in those packets.

TCP/IP has two main protocols that operate at the transport layer of the OSI Reference Model. One is TCP, which is connection-oriented; the other, the User Datagram Protocol (UDP), is connectionless. TCP is used for applications that require the establishment of connections (as well as TCP's other service features), such as FTP; it works using a set of rules, as described earlier, by which a logical connection is negotiated prior to sending data. UDP is used by other applications that don't need connections or other features, but do need the faster performance that UDP can offer by not needing to make such connections before sending data.

Some people consider the layering of a connection-oriented protocol over a connectionless protocol to be like a simulation of circuit switching at higher network layers; this is perhaps a dubious analogy. Even though you can use a TCP connection to send data back and forth between devices, all that data is indeed still being sent as packets; there is no real circuit between the devices. This means that TCP must deal with all the potential pitfalls of packet-switched communication, such as the potential for data loss or receipt of data pieces in the incorrect order. Certainly, the existence of connection-oriented protocols like TCP doesn't obviate the need for circuit-switching technologies, though you will get some arguments about that one too.

The principle of layering also means that there are other ways that connection-oriented and connectionless protocols can be combined at different levels of an internetwork. Just as a connection-oriented protocol can be implemented over an inherently connectionless protocol, the reverse is also true: a connectionless protocol can be implemented over a connection-oriented protocol at a lower level. In a preceding example, I talked about Telnet (which requires a connection) running over IP (which is connectionless). In turn, IP can run over a connection-oriented protocol like Asynchronous Transfer Mode (ATM).

## **Messages: Packets, Frames, Datagrams, and Cells**

Many networking technologies are based on packet switching, which involves the creation of small chunks of data to be sent over a network. Even though *packet* appears in the name of this method, the data items sent between networked devices are most generically called *messages*. *Packet* is one of a variety of similar words that are used in different contexts to refer to messages sent from one device to another.

In some cases, the different terms can be very useful, because the name used to refer to a particular message can tell you something about what the message contains, as you will see shortly. In particular, different message names are usually associated with protocols and technologies operating at specific layers of the OSI Reference Model. Thus, the use of these different names can help clarify discussions that involve multiple protocols operating at different layers.

Unfortunately, these terms can also cause confusion, because they are not always applied in a universal or even consistent manner. Some people are strict about applying particular message designations only to the appropriate technologies

where they are normally used, while others use the different terms completely interchangeably. This means that you should be familiar with the different message types and how they are normally used, but you should still be prepared for the unexpected.

The most common terms used for messages are the following:

**Packet** This term is considered by many to correctly refer to a message sent by protocols operating at the network layer of the OSI Reference Model. So you will commonly see people refer to *IP packets*. However, this term is commonly also used to refer generically to any type of message, as I mentioned earlier.

**Datagram** This term is basically synonymous with *packet* and is also used to refer to network layer technologies. It is also often used to refer to a message that is sent at a higher level of the OSI Reference Model (more often than *packet* is).

**Frame** This term is most commonly associated with messages that travel at low levels of the OSI Reference Model. In particular, it is most commonly seen used in reference to data link layer messages. It is occasionally also used to refer to physical layer messages, when message formatting is performed by a layer 1 technology. A frame gets its name from the fact that it is created by taking higher-level packets or datagrams and “framing” them with additional header information needed at the lower level.

**Cell** Frames and packets, in general, can be of variable length, depending on their contents; in contrast, a *cell* is most often a message that is fixed in size. For example, the fixed-length, 53-byte messages sent in ATM are called cells. Like frames, cells are usually used by technologies operating at the lower layers of the OSI model.

**Protocol Data Unit (PDU) and Service Data Unit (SDU)** These are the formal terms used in the OSI Reference Model to describe protocol messages. A PDU at layer N is a message sent between protocols at layer N. It consists of layer N header information and an encapsulated message from layer N+1, which is called both the *layer N SDU* and the *layer N+1 PDU*. After you stop scratching your head, see the “Data Encapsulation, Protocol Data Units (PDUs), and Service Data Units (SDUs)” section in Chapter 5 for a discussion of this.

I should also point out that there are certain protocols that use unusual names, which aren’t used elsewhere in the world of networking, to refer to their messages. One prominent example is TCP, which calls its messages *segments*.

**KEY CONCEPT** Communication between devices on packet-switched networks is based on items most generically called messages. These pieces of information also go by other names such as packets, datagrams, frames, and cells, which often correspond to protocols at particular layers of the OSI Reference Model. The formal OSI terms for messages are *protocol data unit (PDU)* and *service data unit (SDU)*.

In this book, I have made a specific effort not to imply anything about the nature of a message solely based on the name it uses, but I do follow the most common name used for a particular technology. For example, messages sent over Ethernet

are almost always called Ethernet frames—they are not generally called Ethernet datagrams, for example. However, I do not structure discussions so that the type of name used for a message is the only way to determine what sort of message it is.

## **Message Formatting: Headers, Payloads, and Footers**

*Messages* are the structures used to send information over networks. They vary greatly from one protocol or technology to the next in how they are used, and as just described, they are also called by many different names. Shakespeare had the right idea about names, however. The most important way that messages differ is not in what they are called but in terms of their *content*.

Every protocol uses a special *formatting method* that determines the structure of the messages it employs. Obviously, a message that is intended to connect a web server and a web browser is going to be quite different from one that connects two Ethernet cards at a low level. This is why I separately describe the formats of dozens of different protocol messages in various areas of this book.

While the format of a particular message type depends entirely on the nature of the technology that uses it, messages on the whole tend to follow a fairly uniform overall structure. In generic terms, each message contains the following three elements (see Figure 1-3):

**Header** Information that is placed before the actual data. The header normally contains a small number of control-information bytes, which are used to communicate important facts about the data that the message contains and how it is to be interpreted and used. It serves as the communication and control link between protocol elements on different devices.

**Data** The actual data to be transmitted, often called the *payload* of the message (metaphorically borrowing a term from the space industry!). Most messages contain some data of one form or another, but some messages actually contain none. They are used for only control and communication purposes. For example, these may be used to set up or terminate a logical connection before data is sent.

**Footer** Information that is placed after the data. There is no real difference between the header and the footer, as both generally contain control fields. The term *trailer* is also sometimes used.



**Figure 1-3: Network message formatting** In the most general of terms, a message consists of a data payload that will be communicated, bracketed by a set of header and footer fields. The data of any particular message sent in a networking protocol will itself contain an encapsulated higher-layer message containing a header, data, and a footer. This “nesting” can occur many times as data is passed down a protocol stack. The header is found in most protocol messages; the footer only in some.

Since the header and footer can contain both control and information fields, you might rightly wonder what the point is of having a separate footer anyway. One reason is that some types of control information are calculated using the values of the data itself. In some cases, it is more efficient to perform this computation as the data payload is being sent, and then transmit the result after the payload in a footer. A good example of a field often found in a footer is redundancy data such as cyclic redundancy check (CRC) code, which can be used for error detection by the receiving device. Footers are most often associated with lower-layer protocols, especially at the data link layer of the OSI Reference Model.

**KEY CONCEPT** The general format of a networking message consists of a *header*, followed by the *data* or *payload* of the message, followed optionally by a *footer*. Header and footer information is functionally the same except for its position in the message; footer fields are only sometimes used, especially in cases where the data in the field is calculated based on the values of the data being transmitted.

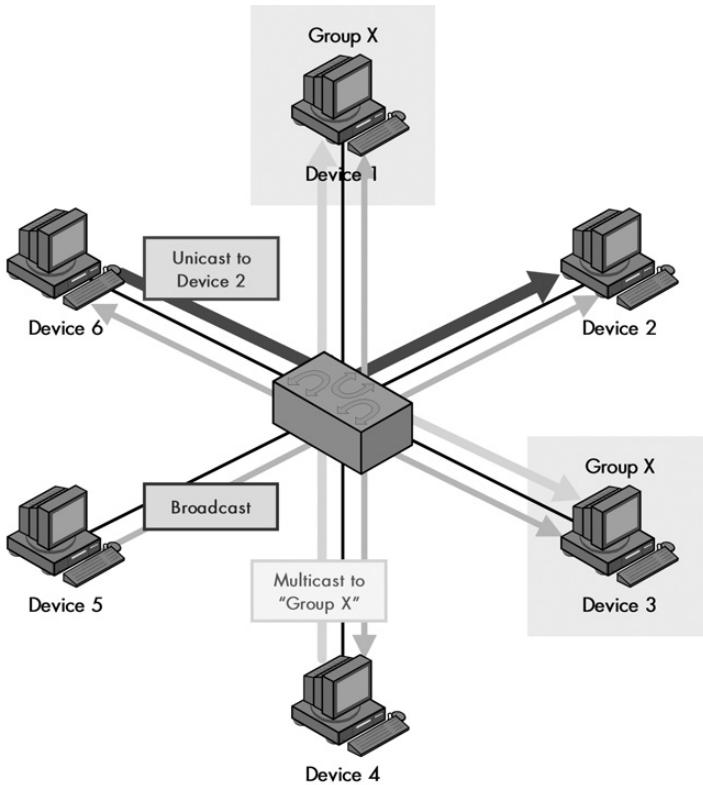
Generally speaking, any particular protocol is concerned with only its *own* header (and footer, if present). It doesn't care much about what is in the data portion of the message, just as a delivery person worries only about driving the truck and not so much about what it contains. At the beginning of that data will normally be the headers of other protocols that were used higher up in the protocol stack; this, too, is shown in Figure 1-3. In the OSI Reference Model, a message handled by a particular protocol is said to be its PDU; the data it carries in its payload is its SDU. The SDU of a lower-layer protocol is usually a PDU of a higher-layer protocol. The discussion of data encapsulation in Chapter 5 contains a full explanation of this important concept.

### ***Message Addressing and Transmission Methods: Unicast, Broadcast, and Multicast***

In a networking technology that uses messages to send data, you must undertake a number of tasks in order to successfully transmit the data from one place to another. One is simply *addressing* the message—putting an address on it so that the system knows where it is supposed to go. Another is *transmitting* the message, which is sending it to its intended recipient.

There are several different ways of addressing and transmitting a message over a network. One way in which messages are differentiated is in how they are addressed and how many recipients will receive them. The method used depends on the function of the message and also on whether or not the sender knows specifically or generally whom they are trying to contact.

To help explain these different methods, I will use a real-world analogy. Consider a social function with 300 people that is being held in a large hall. These people are mingling and having different conversations. There are different kinds of messages that you may need to send in this setting, as is the case with networks.



**Figure 1-4: Unicast, multicast, and broadcast message addressing and transmission** The three basic types of addressing and message delivery in networking are illustrated in this simplified LAN. Device 6 is sending a unicast message to Device 2, shown as the dark, heavy arrow. Device 4 is sending a multicast message to multicast group X, shown as the medium-weight arrows. In this case, that group includes Devices 1 and 3, which are highlighted. Finally, Device 5 is sending a broadcast message, which goes to all other devices on the LAN, shown as the thin, faint arrows.

Bearing this analogy in mind, consider these three kinds of message transmissions, which are illustrated in Figure 1-4:

**Unicast Messages** These are messages that are sent from one device to another device; they are not intended for others. If you have a friend at this social event, this is the equivalent of pulling him aside for a private conversation. Of course, there is still the possibility of someone else at the event overhearing your conversation—or even eavesdropping on it. The same is true in networking as well—addressing a message to a particular computer doesn’t guarantee that others won’t also read it; it’s just that they normally will not do so.

**Broadcast Messages** As the name suggests, these messages are sent to every device on a network. You use them when you need to communicate a piece of information to everyone on the network, or when the sending station needs to send it to just one recipient, but doesn’t know its address. For example, suppose a new arrival at the social gathering saw in the parking lot a blue sedan with its lights left on. She does not know who the car belongs to. The best way to communicate this

information is to broadcast it by having the host make an announcement that will be heard by all, including the vehicle’s owner. In networks, broadcast messages are used for a variety of purposes, including finding the locations of particular stations or the devices that manage different services.

**Multicast Messages** These are a compromise between the previous two types. Multicast messages are sent to a group of stations that meet a particular set of criteria. These stations are usually related to each other in some way. For example, they serve a common function or are set up into a particular *multicast group*. (Note that you can also consider broadcast messages to be a special case of multicast in which the group is “everyone.”)

Back to our analogy: This would be somewhat like a group of friends who go to this large social hall and then stay together in a small discussion group—or perhaps use radios to talk to each other from a distance. Multicasting requires special techniques that make clear who is in the intended group of recipients.

Since these transmission methods differ based on how many and which devices receive the transmission, they are tied directly to the methods used for addressing, as follows:

**Unicast Addressing** Unicast delivery requires that a message should be addressed to a specific recipient. This is the most common type of messaging, so this addressing capability is present in almost all protocols.

**Broadcast Addressing** Broadcasts are normally implemented via a special address that is reserved for that function. Whenever devices see a message sent to that address, they all interpret it as “This message goes to everyone.”

**Multicast Addressing** Multicasts are the most complex type of message because they require a means of identifying a set of specific devices that will receive a message. It is often necessary to create several such groups, which may or may not partially overlap in their membership. Some mechanism is needed to manage which devices are in which groups.

**KEY CONCEPT** Three basic methods are used to address and transmit data between networked devices. A *unicast* transmission goes from one device to exactly one other; this is the most common method used for most message transactions. A *broadcast* transmission is sent from one device to all connected devices on a network. A *multicast* transmission is addressed and sent to a select group of devices.

Finally, one special case in the field of addressing is worth mentioning. In some networks or links, only two devices are connected together, forming what is often called a *point-to-point network*. In this situation, everything sent by one device is implicitly intended for the other, and vice versa. Thus, no addressing of messages on a point-to-point link is strictly necessary.

**NOTE** A new type of message-addressing method was defined as part of IP version 6 (IPv6): the *anycast* message. This term identifies a message that should be sent to the closest member of a group of devices. Chapter 25 describes this type of addressing and transmission.

## **Network Structural Models and Client-Server and Peer-to-Peer Networking**

I mentioned in my discussion of the advantages of networking that networks are normally set up for two primary purposes: *connectivity* and *sharing*. If you have a network with a number of different machines on it, each computer can interact with another's hardware and software, which enables you to perform a variety of tasks. How this is actually done depends to a large degree on the overall design of the network.

One very important issue in network design is how to configure the network for the sharing of resources. Specifically, the network designer must decide whether or not to dedicate resource management functions to the devices that constitute it. In some networks, all devices are treated equally in this regard, while in others, each computer is responsible for a particular job in the overall function of providing services. In this latter arrangement, the devices are sometimes said to have *roles*, somewhat like actors in a play.

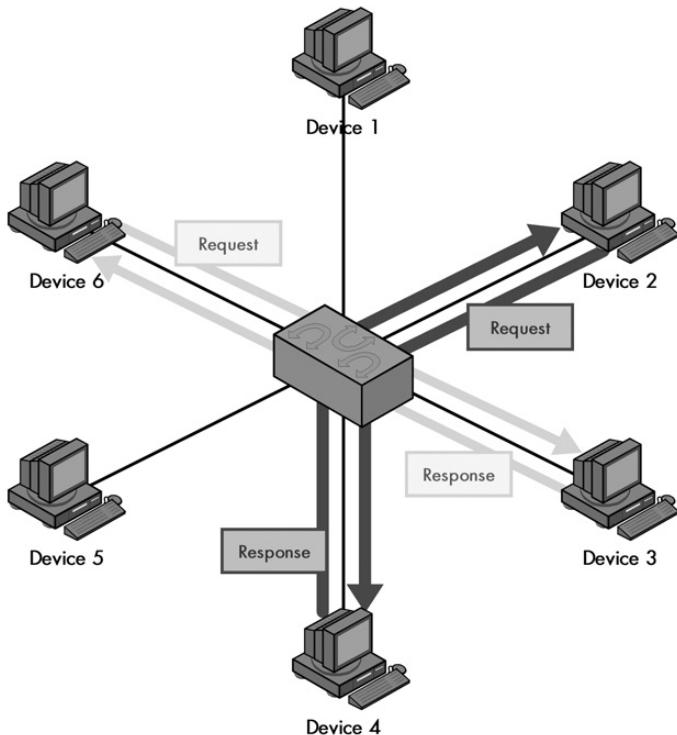
The following two common terms are used to describe these different approaches to setting up a network:

**Peer-to-Peer Networking** In a strict peer-to-peer networking setup, every computer is an equal, a *peer* in the network. Each machine can have resources that are shared with any other machine. There is no assigned role for any particular device, and each of the devices usually runs similar software. Any device can and will send requests to any other, as illustrated in Figure 1-5.

**Client-Server Networking** In this design, a small number of computers are designated as centralized *servers* and are given the task of providing services to a larger number of user machines called *clients*, as shown in Figure 1-6. The servers are usually powerful computers with a lot of memory and storage space, and fast network connections. The clients are typically smaller, regular computers like PCs; they are optimized for human use.

The term *client-server* also frequently refers to protocols and software, which are designed with matching, complementary components. Usually, server software runs on server hardware, and client software is used on client computers that connect to those servers. Most of the interaction on the network is between client and server, not between clients. Server software is designed to efficiently respond to requests, while client software provides the interface to the human users of the network.

**KEY CONCEPT** Networks are usually configured to share resources using one of two basic *structural models*. In a *peer-to-peer network*, each device is an equal, and none are assigned particular jobs. In a *client-server network*, however, devices are assigned particular roles—a small number of powerful computers are set up as *servers* and respond to requests from the other devices, which are *clients*. Client-server computing also refers to the interaction between complementary protocol elements and software programs. It's rising in popularity due to its prevalence in TCP/IP and Internet applications.



**Figure 1-5: Peer-to-peer networking** In this model, each device on the network is treated as a peer, or equal. Each device can send requests and responses, and none are specifically designated as performing a particular role. This model is more often used in very small networks. Contrast this with Figure 1-6.

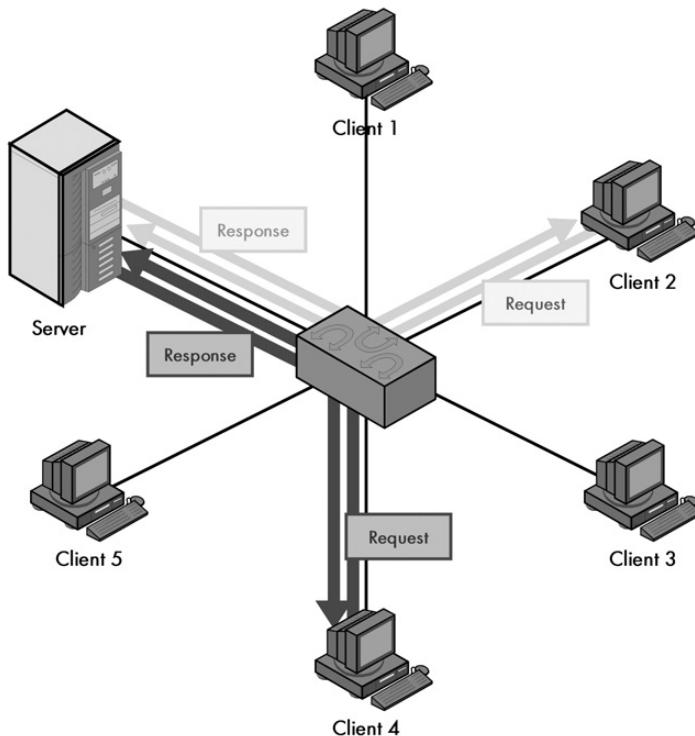
The choice of client-server or peer-to-peer is sometimes called choosing a *structural model* for the network. As with most situations in which two different schemes are used, there is no right answer in this regard. Your choice depends entirely on the needs of the particular network.

Peer-to-peer networking has primary advantages of simplicity and low cost, meaning that it has traditionally been used on small networks. Client-server networking provides advantages in the areas of performance, scalability, security, and reliability, but is more complicated and expensive to set up. This makes it better suited to larger networks. Over time, however, there has been a steady evolution toward client-server networking, even on smaller networks. Many years ago, it was common to see even networks with 20 to 50 machines using the peer-to-peer model; today, even networks with only a half-dozen machines are sometimes set up in a client-server mode because of the advantages of centralized resource serving.

The rise in popularity of client-server networking is ironic because, in some ways, it is actually a throwback to the days of large mainframes decades ago. A mainframe with attached terminals can be thought of as a client-server network, with the mainframe itself being the server and the terminals being clients. This analogy is not perfect, of course, because modern client computers do a lot more work than dumb terminals on mainframes.

One of the reasons why the client-server structural model is becoming dominant is that it is the primary model used by the world's largest network: the Internet. Client-server architecture is the basis for most TCP/IP protocols and services. For example, the term *web browser* is really another name for a web client, and a *website* is really a web server.

**NOTE** For more information on client-server computing, I recommend that you read the section "TCP/IP Services and Client/Server Operation" in Chapter 8. That topic also contains a very relevant exposition on the different meanings of the terms client and server in hardware, software, and transactional contexts.



**Figure 1-6: Client-server networking** In the client-server model, a small number of devices are designated as servers and equipped with special hardware and software that allows them to interact efficiently and simultaneously with multiple client machines. Though the clients can still interact with each other, most of the time they send requests of various sorts to the server, and the server sends back responses to them. Contrast this with the peer-to-peer networking example in Figure 1-5.

## Types and Sizes of Networks

One of the reasons that understanding networks can be difficult at times is that there are so many different types! When someone talks about a network, she can mean anything from two computers hooked together in an apartment to a globe-spanning entity with millions of nodes. Every network is unique, and each one has

an important role to play in filling the communication and data-sharing needs of different individuals and organizations. In fact, the great diversity and flexibility of networking is one of its most important strengths.

Two of the most basic ways that you can distinguish and contrast various networks are the relative distances between the devices that they connect and the general mechanisms used to communicate between them. The reason for making these distinctions is that the technological needs of a network differ greatly depending on the amount of ground you are trying to cover, and also by the overall way that you want to transmit and receive information.

Many people, including me, like to divide the many kinds of networks in existence into three general classes, as follows:

**Local Area Networks (LANs)** Networks that connect computers that are relatively close to each other—generally, within the same room or building. When most people think about networking PCs and other small computers, this is what they usually have in mind. The vast majority of regular LANs connect using cables, so the term *LAN* by itself usually implies a wired LAN, but not always.

**Wireless LANs (WLANs)** LANs that connect devices without wires, using radio frequencies or light. WLANs can be entirely wireless, but most are not. They usually connect wireless devices to each other as well as to the wired portion of the network. Due to the limits of most wireless technologies, WLANs usually connect devices that are very close to each other, generally within a few hundred feet at most.

**Wide Area Networks (WANs)** Networks that connect devices or other networks over a greater distance than that which is practical for LANs. If the distance between devices can be measured in miles, you will generally use WAN and not LAN technology to link them.

More often than not, WANs are used to link physically distant LANs. For example, a company with locations in two different cities would normally set up a LAN in each building and then connect them together in a WAN. I also consider most Internet access technologies to be a form of WAN, though some might not agree with that. There is also the term *wireless WAN (WWAN)*, which just refers to a WAN that uses wireless technology.

As with most other distinctions and categorizations in the world of networking, the lines between these various definitions are not very concrete. As I mentioned already, WLANs are usually not entirely wireless because they contain wired elements. Similarly, trying to say absolutely when a network is “local” and when it is “wide” is difficult.

It’s also somewhat pointless to spend too much energy on differentiating these network classes precisely. In some cases it’s not the definitions that decide what technology to use, but rather the technology that indicates what kind of network you have! Since some protocols are designed for WANs, if you are using them, many would say you have a WAN, even if all the devices in that technology are near each other. On the other hand, some LAN technologies allow for the use of cables that can run for many miles; most would still consider a mile-long Ethernet fiber link to be a LAN connection, even though it may span WAN distances.

There are many dimensions in which LAN and WAN technologies differ; two of the most important are *cost* and *performance*. It's easy to establish a high-speed conduit for data between two systems that are in the same room, but it's much more difficult if the two are in different states. This means that in the world of WAN, one either pays a lot more or gets a lot less throughput—often it's both.

The gray area between LAN and WAN is becoming more muddled every year. One reason is the emergence of intermediate network types that straddle the line between these more familiar terms. Two of the more common ones are as follows:

**Campus Area Networks (CANs)** A *CAN* is one created to span multiple buildings in the same location, such as the campus of a university. Campus area networking is a gray area, since neither LANs nor WANs alone are always well suited for this type of application. Often, a mix of LAN and WAN techniques is used for campus networking, depending on the characteristics of the campus and the needs of the organization.

**Metropolitan Area Networks (MANs)** Another intermediate term that you may see sometimes is the *MAN*. As the name implies, this refers to a network that spans a particular small region or a city. MANs can be considered small WANs that cover a limited geographical area, or large LANs that cover an area greater than what is normally associated with a local network. Wireless MANs are sometimes called WMANs; IEEE 802.16 is an example of a WMAN standard.

Finally, there is one other term occasionally used that I should mention: the *personal area network (PAN)*. This type of network generally means a very small LAN with a range of only a few feet. PANs mostly connect devices used by a single person (or very small group). The term is most commonly used in reference to Bluetooth/IEEE 802.15 wireless technology, so you will sometimes see the terms *wireless PAN (WPAN)* and *PAN* used interchangeably.

**KEY CONCEPT** Networks are often divided by size and general communication method into three classes. *Local area networks (LANs)* generally connect proximate devices, usually using cables. *Wireless LANs (WLANs)* are like cabled LANs but use radio frequency or light technology to connect devices without wires. *Wide area networks (WANs)* connect distant devices or LANs to each other. *Campus area networks (CANs)* and *metropolitan area networks (MANs)* fall between LANs and WANs in terms of overall size. *Personal area networks (PANs)* are like very small LANs and often appear as *wireless PANs (WPANs)*.

## Segments, Networks, Subnetworks, and Internetworks

One of the reasons that networks are so powerful is that they can be used to connect not only individual computers, but also groups of computers. Thus, network connections can exist at multiple levels; one network can be attached to another network, and that entire network can be attached to another set of networks, and so on. The ultimate example of this is, of course, the Internet, which is a huge collection of networks that have been interconnected into . . . dare I say, a web?

This means that a larger network can be described as consisting of several smaller networks or even parts of networks that are linked together. Conversely, we can talk about taking individual networks or network portions and assembling them into larger structures. The reason why this concept is important is that certain technologies are best explained when looking at an entire large network at a high level, while others really require that you drill down to the detailed level of how constituent network pieces work.

Over time, a collection of terms has evolved in the networking world to describe the relative sizes of larger and smaller networks. Some of the most common ones are as follows:

**Network** This is the least specific of the terms mentioned here. Basically, a *network* can be pretty much any size, from two devices to thousands. When networks get very large, however, and are clearly comprised of smaller networks connected together, they are often no longer called networks but *internetworks*, as you will see momentarily. Despite this, it is fairly common to hear someone refer to something like “Microsoft’s corporate network,” which obviously contains thousands or even tens of thousands of machines.

**Subnetwork (Subnet)** A *subnetwork* is a portion of a network, or a network that is part of a larger internetwork. This term is also a rather subjective one; subnetworks can be rather large when they are part of a network that is very large.

The abbreviated term *subnet* can refer generically to a subnetwork, but also has a specific meaning in the context of TCP/IP addressing (see Chapter 18).

**Segment (Network Segment)** A *segment* is a small section of a network. In some contexts, a segment is the same as a subnetwork and the terms are used interchangeably. More often, however, the term *segment* implies something smaller than a subnetwork. Networks are often designed so that, for the sake of efficiency, computers that are related to each other or that are used by the same groups of people are put on the same network segment.

Some LAN technologies—including Ethernet—use the term *segment* to refer specifically to a collection of geographically proximate machines that are connected directly to each other, either by a single cable or single device such as a hub. Such technologies have specific rules about how many devices can be on a segment, how many segments can be connected together, and so on, depending on what sort of network interconnection devices you are using.

**Internetwork (or Internet)** Most often, this refers to a larger networking structure that is formed by connecting smaller ones. Again, the term can have either a generic or a specific meaning, depending on context. In some technologies, an internetwork is just a very large network that has networks as components. In others, a network is differentiated from an internetwork based on how the devices are connected together.

An important example of the latter definition is TCP/IP, in which a *network* usually refers to a collection of machines that are linked at layer 2 of the OSI Reference Model, using technologies like Ethernet or Token Ring, as well as interconnection devices such as hubs and switches. An internetwork is formed when these networks are linked together at layer 3, using routers that pass IP datagrams

between networks. Naturally, this is highly simplified, but in studying TCP/IP, you should keep this in mind when you encounter the terms *network* and *internetwork*.

**NOTE** *The shorter form of the word internetwork (internet) is often avoided by people who wish to avoid confusion with the proper noun form (Internet). The latter, of course, refers only to the well-known global internetwork of computers and all the services it provides. I personally try to use the word internetwork most of the time in this book instead of internet, for this very reason.*

Understanding these different terms is important not only for helping you comprehend what you read about networks, but also because they are important concepts in network design. This is particularly true for LANs in which decisions regarding how to set up segments and how to connect them to each other have an important impact on the overall performance and usability of the network.

**KEY CONCEPT** Several terms are often used to describe the relative sizes of networks and parts of networks. The most basic term is *network* itself, which can refer to most anything, but often means a set of devices connected using an OSI layer 2 technology. A *subnetwork* is a part of a network (or internetwork), as is a *segment*, though the latter often has a more specific meaning in certain technologies. An *internetwork* refers either generically to a very large network, or specifically, to a set of layer 2 networks connected using routers at layer 3.

The term *segment* is notably problematic because it is routinely used in two different ways, especially in discussions related to Ethernet. Traditionally, a *segment* referred to a specific cable. The earliest forms of Ethernet used coaxial cables, and the coaxial cable segment was shared and became the collision domain for the network. *Collision domain* is a term that refers generally to a collection of hardware devices in which only one can transmit at a time. Devices such as hubs and repeaters were used to extend collision domains by connecting together these segments of cable into wider networks. Over time, the terms *collision domain* and *segment* started to be used interchangeably. Thus today a segment can refer either to a specific piece of cable or to a collection of cables connected electrically that represent a single collision domain.

**NOTE** *As if that potential ambiguity in the use of the word segment isn't bad enough, it also has another, totally unrelated meaning: It is the name of the messages sent in TCP!*

## The Internet, Intranets, and Extranets

I mentioned in the preceding discussion of segments, networks, subnetworks, and internetworks that the Internet is really the king of internetworks. After all, you don't get to be called "the" something unless you pretty much define it!

In fact, the Internet is not just a large internetwork, but substantially more. The Internet is defined not just as the computers that are connected to each other around the world, but as the set of services and features that it offers. More than that, the Internet defines a specific way of doing things, of sharing information and resources between people and companies. And though it might be a bit melodramatic to say so, to many people, the Internet is a way of life.

As Internet use exploded in the 1990s, many people realized that the techniques and technologies used on the Internet would be useful if applied to internal company networks as well. The term *intranet* was coined to refer to an internal network that functioned like a private Internet. It comes from the prefix *intra*, which means within. Of course, *inter* is the opposite of intra, so this makes some people think that an intranet is the opposite of an internet. In fact, most intranets *are* internetworks as well!

As if that weren't bad enough from a jargon standpoint, the buzzword buzzards then decided to take matters a step further. If an intranet is extended to allow access to it not only by people or groups strictly from within the organization, but also by people or groups outside the main company, this is sometimes called an *extranet*. *Extra*, of course, is a prefix that means outside, or beyond.

So, an extranet is a type of internal, private Internet that, well, isn't entirely internal. An extranet is an extended intranet, which is really a type of internet that works like the Internet. (You can start to see why I am not a big fan of these fancy terms. But then, I don't get to choose them; I just have to help you understand them!) An extranet isn't public and open to all—it is controlled by a private organization. At the same time, it isn't entirely private either.

**KEY CONCEPT** The generic noun *internet* is a short form for the word internetwork, while the proper noun *Internet* refers to the global internetwork of TCP/IP networks that we all know and use. The term *intranet* refers to an internal network that uses TCP/IP technologies as the Internet does. An *extranet* is like an intranet that is extended to individuals or organizations outside the company. All these terms can be used ambiguously, so you must take care to determine exactly what they mean in any given context.

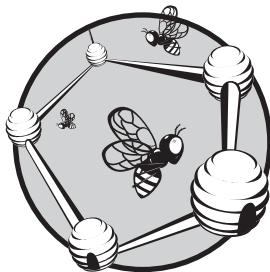
As you can see, the lines between the Internet, intranets, and extranets were pretty blurry from the start, and the concepts are rapidly blending into a diffuse gray mass as the whole computing world becomes more tightly integrated. For example, even if you have an entirely private intranet, you will want to connect it to the Internet to communicate with the outside world and to allow access to Internet resources. And an extranet may be implemented, in part, through the public Internet infrastructure, using technologies such as virtual private networking (VPN). I think you get the picture.

The key that binds all of these concepts together is that they all use *Internet technologies*, which is a term that is also somewhat vague. This usually refers to the use of the TCP/IP protocol suite, which is the defining technology of the Internet as well as the set of services that are available on the Internet.

The bottom line is that being told that a company has an intranet or an extranet—as opposed to a plain old boring network—doesn't tell you much at all. It is best not to rely on the slogans and instead look at the underlying characteristics of the network or internetwork itself. Furthermore, when designing such a network, you should focus on using the technologies and protocols that make sense—let the marketing people decide what to call it later.

# 2

## NETWORK PERFORMANCE ISSUES AND CONCEPTS



Networking is largely about connecting together devices so that information can be shared between them. Since the idea is to send data from one place to another, a very important characteristic of any network is its *speed*: How fast can data be transmitted and received? This matter of speed turns out to be only one of several issues that determine the overall *performance* of a network.

In the computing world, performance is, in general, one of the most discussed but least understood characteristics of any system or hardware device. This is true of networking as well. For example, most people know the raw throughput rating of their network hardware, and they may even start to draw conclusions about its capabilities based on those numbers. Many, however, don't realize that they will never actually achieve that rated amount of performance in the real world.

Most of the other subtle issues related to performance are also typically ignored or misunderstood, such as the impact of software drivers on hardware performance, and the fact that certain applications need more than just raw bit speed—they need *reliable* delivery of data. But even beyond all of this, one of the most important issues related to network performance is understanding what your needs are, because then you can make sure you don't spend too much money for performance you don't need—or worse, create a network that can't meet your requirements.

In this chapter, I discuss various performance issues and concepts that are related to networking in one way or another. First and foremost, I try to put performance in context and also contrast it with nonperformance issues. Then I talk about several key performance terms and metrics: speed, bandwidth, throughput, and latency. I also discuss some of the units used to measure network performance. I then explain how the real-world performance of a network differs from its theoretical performance and talk about factors that have an impact on network performance. I conclude by contrasting full-duplex and half-duplex network operation, and talking about quality of service, which is a concept that is especially important in the use of networks for real-time applications such as streaming multimedia.

## Putting Network Performance in Perspective

*Performance* is probably the mother of all buzzwords in the computer industry. There are many people who consider it the ultimate goal of any computer or computer system, and by extension, any network. A lot of people spend many dollars and hours of time trying to maximize it. There's good reason for this: Performance is very important. A network that does not offer adequate performance simply will not get the job done for those that rely on it. However, it is very important to keep performance in perspective. Successfully maximizing performance requires that you first take a step back and look at the big picture.

The first question you must ask yourself is also the most essential one: How important is performance to you? Before you answer this question, recall the old auto racing adage: “Speed costs money—how fast do you want to go?” While there are some situations in which you can get much better performance in a network by spending just a little more money, in general, you don't get more performance without paying for it in some way. That may mean a higher cost for the network, or it may mean a trade-off of some nonperformance feature.

If you are designing or specifying a network, it's very important to keep in mind that your goal is to come up with a system that will meet the needs that were determined for it during requirements analysis. This means coming up with a network that has a level of performance that matches the requirements and leaves some room for expansion. Unless you have an unlimited budget—and who does, right?—your objective is not “performance at any cost.” It is to create a network that meets *all* of your users' needs, including balancing performance and nonperformance characteristics, as you will see shortly.

**NOTE** *Buyer beware. Companies are constantly coming out with the “latest and greatest” high-performance networking technologies. They usually try to sell their technologies by attempting to convince you that you just cannot live without this latest advance; that you “need” to upgrade—*

*immediately, if not sooner! Well, it's simply not the case. For example, even though you can find Ethernet hardware that runs with a theoretical throughput of 10,000 megabits per second, there are many older networks that continue to work just fine at one 1/100th that speed—or even 1/1,000th!*

Finally, remember that designing a network is usually not an irreversible, permanent decision. Networks can be upgraded and expanded. While it is prudent to build some slack into any network to allow for growth, it is not wise to spend too much time and money planning for the future when changes can be made later. This is especially true given that network hardware prices drop over time. Again, this is a matter of drawing an appropriate balance between future performance planning and budget.

## Balancing Network Performance with Key Nonperformance Characteristics

We all know that performance is very important to any network. However, anyone putting together a network must also be concerned with many different nonperformance characteristics as well. Depending on the network, these can be just as essential to the users of the network as performance, and possibly even more critical. More than this, nonperformance issues often *trade off* against performance, and in fact, more often than not they have to be reduced to get performance to increase.

Before you can create a very high-performance network, you must understand the key nonperformance network characteristics that you may need to compromise. The following are a few of these issues:

**Design and Implementation Cost** Unless you have bottomless pockets, you must be concerned with the network's costs. As mentioned earlier, cost is the main trade-off with performance. Going faster usually costs more money.

**Quality** The quality of the network is a function of the quality of the components used and how they are installed. Quality is important because it impacts all of the other factors, such as reliability and ease of administration, as well as performance. Quality doesn't trade off *directly* with performance—you can design high-quality, high-performance networks—but it does *compete* with performance for resources in the budget. All else being equal, it costs a great deal more to implement a high-quality, high-performance network than a high-quality, low-speed one.

**Standardization** Network protocols and hardware can be designed to meet either universally accepted standards or nonstandard, proprietary ones. Standard designs are almost always preferable, because they make interoperability, upgrading, support, and training easier. Proprietary standards may include enhancements that improve performance, but may increase cost or make management more difficult.

**Reliability** This is related to several other issues, especially quality and performance. Faster networks aren't necessarily less reliable, but they are more difficult and expensive to make run reliably than slower ones.

**Expandability and Upgradability** It's very important to always plan for the future when creating a network. Higher-performance networks can be more difficult to expand, and they are certainly more expensive to expand. Once again, the matter of implementing a network with a capacity for future needs now, as opposed to upgrading later if it becomes necessary, is an important network design decision.

**Ease of Administration and Maintenance** Higher-performance networks require more work and resources to administer and maintain, and they are more likely to require troubleshooting than slower ones.

**Premises and Utility Issues** Implementation of high-speed networks may be limited by the physical premises or may have an impact on how they are laid out. Choosing a higher-speed option may require more infrastructure to be put in place, thus increasing cost. The classic example of this is choosing between wired and wireless options for a home or small office network. You can go much faster with wires, but do you really want to run them?

**KEY CONCEPT** While performance is one of the most important characteristics of any network, there are others that are equally important. In many cases, you must weigh the cost, quality, reliability, expandability, maintainability, and other attributes of a network against overall performance. The faster you want your network to go, the more difficult it is to ensure that these other attributes are kept at sufficiently high levels.

## Performance Measurements: Speed, Bandwidth, Throughput, and Latency

A number of terms are commonly used to refer to various aspects of network performance. Some of them are quite similar to each other, and you will often see them used—and in many cases, misused or even *abused*! I'll examine each of them to see how they are commonly used and what they really mean.

**NOTE** *More than just the issue of different terms related to performance, however, is the more important reality that there are multiple facets to performance. Depending on the application, the manner in which data is sent across the network may be more important than the raw speed at which it is transported. In particular, many multimedia applications require real-time performance; they need data sent in such a manner that it will be delivered steadily. For these purposes, raw speed isn't as important as consistent speed. This is an issue that is often not properly recognized.*

### Speed

*Speed* is the most generic performance term used in networking. As such, it can mean just about *anything*. Most commonly, however, it refers to the *rated* or *nominal* speed of a particular networking technology. For example, Fast Ethernet has a nominal speed of 100 Mbps (megabits per second); for that reason, it is often called 100 Mbps Ethernet, or given a designation such as 100BASE-TX.

Rated speed is the biggest performance magic number in networking—you see it used to label hardware devices, and many people bandy the numbers about as if those numbers actually represented the network's real speed. The problem with

using nominal speed ratings is that they are only *theoretical*, and as such, tell an incomplete story. No networking technology can run at its full-rated speed, and many run *substantially* below it, due to real-world performance factors.

Speed ratings such as 100 Mbps Ethernet are also often referred to as the *throughput* of a technology, even though the maximum theoretical speed of a technology is more analogous to bandwidth than throughput, and the two are not identical.

## ***Bandwidth***

Bandwidth is a widely used term that usually refers to the data-carrying capacity of a network or data-transmission medium. It indicates the maximum amount of data that can pass from one point to another in a unit of time. The term comes from the study of electromagnetic radiation, where it refers to the width of a band of frequencies used to carry data. It is usually given in a theoretical context, though not always.

Bandwidth is still used in these two senses: frequency bandwidth and data capacity. For example, radio frequencies are used for wireless technologies, and the bandwidth of such technologies can refer to how wide the radio frequency band is. More commonly, though, bandwidth refers to how much data can be sent down a network, and it is often used in relative terms. For example, for Internet access, a cable or xDSL connection is considered high-bandwidth access; using a regular analog modem is low-bandwidth access.

## ***Throughput***

Throughput is a measure of how much actual data can be sent per unit of time across a network, channel, or interface. While throughput can be a theoretical term like bandwidth, it is more often used in a practical sense—for example, to measure the amount of data actually sent across a network in the real world. Throughput is limited by bandwidth, or by rated speed: If an Ethernet network is rated at 100 Mbps, that's the absolute upper limit on throughput, even though you will normally get quite a bit less. So, you may see someone say that they are using a 100 Mbps Ethernet connection but getting throughput of, say, 71.9 Mbps on their network.

The terms *bandwidth* and *throughput* are often used interchangeably, even though they are really not exactly the same.

## ***Latency***

*Latency* is a very important, often overlooked term, which refers to the *timing* of data transfers on a communications channel or network. One important aspect of latency is how long it takes from the time a request for data is made until it starts to arrive. Another aspect is how much control a device has over the timing of the data that is sent, and whether the network can be arranged to allow for the consistent delivery of data over a period of time. Low latency is considered better than high latency.

## **Summary of Performance Measurements**

As with all networking terms, there are no hard-and-fast rules; many people are rather loose with their use of terms relating to performance measurement. You will even see terms such as *throughput bandwidth*, *bandwidth throughput*, and other charming inventions from the department of redundancy department. More often, you will just see a lot of mishmashed term usage, and especially, spurious conclusions being drawn about what data streams a network can handle based on its rated speed. Making matters worse is that speed ratings are usually specified in bits per second, but throughput may be given in bits or bytes per second.

**KEY CONCEPT** The three terms used most often to refer to the overall performance of a network are *speed*, *bandwidth*, and *throughput*. These are related and often used interchangeably, but are not identical. The term *speed* is the most generic and often refers to the rated or nominal speed of a networking technology. *Bandwidth* can mean either the width of a frequency band used by a technology or more generally, data capacity, where it's used as more of a theoretical measure. *Throughput* is a specific measure of how much data flows over a channel in a given period of time. It is usually a practical measurement.

In general, *speed*, *bandwidth*, and *throughput* get a lot of attention, while *latency* gets little. Yet latency considerations are very important for many real-time applications such as streaming audio and video and interactive gaming. In fact, they are often more important than raw bandwidth.

For example, suppose you move to a rural home, and your choices for Internet access are a regular 28.8 Kbps modem connection or fancy satellite Internet. The companies selling satellite connectivity call it “broadband” and advertise very high rated speeds—400 Kbps or more. They make a big deal about it being “over ten times as fast as dial-up,” and they certainly charge a lot for this very high-tech service. This is a slam dunk, right?

Wrong. The satellite connection has high bandwidth, but very poor (high) latency due to the time it takes for the signals to travel to and from the satellite. It is definitely much better than the modem for downloading that nice little 150 MB patch from Microsoft. However, it is much *worse* than the modem for playing the latest online video game with your buddy over the Internet, because of the latency, or *lag*, in transmissions. Every move you make in your game will be delayed for over half a second as the signal bounces around between the satellite and the earth, making online gaming nearly impossible. Thus, whether satellite Internet is worth the extra money depends entirely on what you plan to use it for.

**NOTE** An important issue closely related to latency is quality of service, a general term that refers (among other things) to the ability of networks to deliver necessary bandwidth and reliable data transfer for applications that need it. See the section “Quality of Service (QoS)” later in the chapter.

**KEY CONCEPT** Where bandwidth and throughput indicate how fast data moves across a network, *latency* describes the nature of how it is conveyed. It is most often used to describe the delay between the time that data is requested and the time when it arrives. A networking technology with very high throughput and bad (high) latency can be worse for some applications than one with relatively low throughput but good (low) latency.

# Understanding Performance Measurement Units

People who make networking hardware, or write materials that try to tell you how to operate it, use many terms to describe performance, such as *throughput* and *bandwidth*. (These terms are explained in the previous section.) In addition, they also use several different *units* to measure performance. Unfortunately, these units are often used incorrectly, and they are also very similar to each other in name. Worse, they also have overlapping abbreviations, and lots of people use these abbreviations without making clear what the heck they are talking about. Isn't that great?

## Bits and Bytes

The first issue is the infamous letter *B*. Or rather, I should say, the matter of the big *B* and the little *b*. By popular convention, the capitalized *B* is supposed to be used for byte, and the lowercase *b* for bit—this is the way these abbreviations are always used in this book.

**NOTE** *A byte is normally eight bits; sometimes the term octet is used instead. If you aren't familiar with these terms, refer to Chapter 4 for a primer on binary basics, where you will also find a discussion of the small controversy related to bytes and octets.*

Unfortunately, this convention is not followed strictly by everyone. As a result, you may on occasion see *b* being used to refer to bytes, and *B* used for bits. This *b* and *B* business causes a tremendous amount of confusion sometimes, with people mistaking bits for bytes and accidentally thinking that networks are running eight times faster or slower than they really are.

Bear in mind when looking at speed ratings that they are almost always given in terms of bits, not bytes. The 56K in a modem rating means 56,000 bits, not 56,000 bytes of theoretical transfer speed. (This is true even if someone calls it a “56K” modem.) Similarly, Fast Ethernet operates at 100 megabits per second, not megabytes, and a 1.544 Mbps T1 link sends a theoretical maximum of 1,544,000 bits each second. This, at least, is usually pretty consistent.

When it comes to throughput measurements, however, both bits and bytes are used, so you have to be careful. Raw throughput values are normally given in bits per second, but many software applications report transfer rates in bytes per second, including many web browsers and FTP client programs. This often leads to users wondering why they are only getting one-eighth of their expected download or transfer speeds.

**KEY CONCEPT** In most cases in discussions of networking performance, the lowercase letter *b* refers to bits and the uppercase *B* to bytes. However, these conventions are not always universally followed, so context must be used to interpret a particular measurement.

The standard unit for bit throughput is the bit per second, which is commonly abbreviated bit/s, bps, or b/s. The byte unit is byte per second, abbreviated bytes/s, Bps or B/s—unless some cruel author decides to use a lowercase *b* just to confuse you! This means that the maximum theoretical

throughput of 100BASE-TX (100 Mbps) Ethernet is about 12 MB/s. Where the context is unclear, it is better to spell out the unit as 100 Mbits/s or 12 Mbytes/s, which, of course, I try to do in this book.

You will also occasionally, especially when dealing in the realm of communications, see throughput measured in characters per second, or cps. In most computer systems (including PCs), each character takes up one byte, so cps is equivalent to bytes/s, B/s, or Bps.

Of course, most networking technologies don't move just a few bits and bytes around every second; they move, thousands, millions, or even billions. Thus, most speed ratings are not in bits per second, but rather *kilobits* (Kb), *megabits* (Mb), or *gigabits* (Gb) per second, and the same thing can be done for bytes. Thus, you find terms such as 100 Mbps Ethernet or 700 kb/s ADSL.

Here, you run into another problem: the existence of both decimal and binary versions of the terms *kilo*, *mega*, and *giga*. For example, the decimal form of the prefix for a million (mega) is  $10^6$  or 1,000,000, while the binary form is  $2^{20}$  or 1,048,576. This differential of about 5 percent leads to all sorts of confusion. When you see these abbreviations, bear in mind that in networking, they almost always refer to the decimal form. Thus, 100 Mbps Ethernet is rated at 100,000,000 bits per second, not 104,857,600 bits per second.

**KEY CONCEPT** The unit most often used to express networking throughput is *bits per second* or *bps*. This term is often expressed in thousands, millions, or billions as *Kbps*, *Mbps*, or *Gbps*. It almost always uses the decimal, not binary, versions of the *kilo*, *mega*, or *giga* multipliers.

## Baud

Finally, there's another term that you will encounter frequently in discussions of modems and some other technologies: the *baud*. Named for telegraphy pioneer Jean-Maurice-Émile Baudot (1845–1903), this unit measures the number of changes, or transitions, that occur in a signal in each second. So, if the signal changes from a one value to a zero value (or vice versa) one hundred times per second, that is a rate of 100 baud.

In the early days of very slow modems, each bit transition encoded a single bit of data. Thus, 300 baud modems sent a theoretical maximum of 300 bits per second of data. This led to people confusing the terms *baud* and *bits per second*—and the terms are still used interchangeably *far* too often. You will commonly hear people refer to a 28.8 Kbps modem, for example, as running at 28,800 baud.

But the two units are in fact not the same; one measures data (the throughput of a channel), and the other measures transitions (called the *signaling rate*). Modern modems use advanced modulation techniques that encode more than one bit of data into each transition. A 28,800 bps modem typically encodes nine bits into each transition; it runs at 3,200 baud, not 28,800 baud (the latter number being the product of 3,200 and 9). In fact, there's no way to operate a modem on a conventional phone line at 28,800 baud—it exceeds the frequency bandwidth of the phone line. That's the reason why advanced modulation is used to encode more data into each transition.

**KEY CONCEPT** The *baud* and *bps* units are often treated equivalently, but are not the same. *Baud* measures not the throughput of a network but its signaling rate, meaning the number of times that the signal changes value in each second. Since modern encoding and modulation techniques often encode either greater or less than one bit value into each such transition, the throughput and baud rate of network technologies are usually different.

## Theoretical and Real-World Throughput, and Factors Affecting Network Performance

When assessing the performance of networks, keep in mind that there is always a difference between theoretical speed ratings and real-world throughput. If your network is set up well, this difference is relatively small but still significant. Otherwise, the difference can be extremely large. (Notice that the difference between theoretical and practical performance can never be negligible.)

There are many reasons for the difference between what a network or communications method is supposed to be able to do and what it actually can do. The reasons generally fall into three categories: normal network overhead, external performance limiters, and network configuration problems.

**NOTE** *There are many different ways of measuring and assessing performance. Synthetic benchmark programs are often used to measure throughput, and can produce impressive performance scores, which usually have little to do with how a network will actually operate. Such metrics are best used for comparison purposes by showing that one network or system is faster than another, rather than by paying too much attention to the actual number the metrics produce. Even when doing comparisons, however, caution is wise.*

### Normal Network Overhead

Every network has some degree of normal network overhead, which guarantees that you will never be able to use all of the bandwidth of any connection for data. Take as an example 10 Mbps Ethernet. Sure, the line may be able to transmit 10,000,000 bits every second, but not all of those bits are data! Some are used to package and address the data—data can't just be thrown onto the network in raw form. Also, many of those bits are used for general overhead activities, and they deal with collisions on transmissions and other issues. There are natural inefficiencies in any networking technology.

Even beyond this, there are other overhead issues. Any network transaction involves a number of different hardware and software layers, and overhead exists at each of them, from the application and operating system down to the hardware. These overheads mean that you generally lose at least 20 percent of the rated speed of a local area network (LAN) technology off the top, and sometimes even more. For example, 7 Mbps user data throughput on a regular 10 Mbps Ethernet network is actually very good.

## **External Performance Limiters**

There are external factors that limit the performance of a network. Important issues here include the ability of the hardware to process the data and also any bandwidth limitations that exist in the chain of data transmission between two nodes. Hardware issues most often show up with very fast networking technologies.

Consider a gigabit (1,000 Mbps) Ethernet connection between two regular PCs. In theory, this connection should allow the transmission of 1 GB of data every second. Even beyond the matter of overhead mentioned earlier, no regular PC is capable of pumping this much data per second. Only high-end servers have this capacity—and even they would have problems sustaining this unless they were doing nothing else. An older PC's hard disk probably can't even stream data fast enough to keep a 100 Mbps Ethernet connection busy. Thus, upgrading a 100 Mbps Ethernet card in an older machine to gigabit is not likely to help as much as you might expect.

Bandwidth limitations cause network throughput issues because the entire network can run only as fast as its slowest link. These bottlenecks create reduced performance. As a common example, suppose you have a cable modem connection to the Internet that is rated at 1 Mbps for downloads. It may be very fast most of the time, but if the website you are accessing is totally bogged down or it is having connectivity problems itself, you are not going to download from that site at 1 Mbps. In fact, your download probably won't even get close to that speed.

Finally, it's also important to remember that there are many technologies that simply do not always operate at a constant fixed speed, though they may change speeds based on physical network characteristics. A good example is an analog modem, which can vary greatly in performance depending on the quality of the line over which it operates.

## **Network Configuration Problems**

The issues I mentioned earlier are usually ones that you cannot do anything about; they are just the nature of the networking beast. The third category of performance limiters, *misconfiguration*, is different. This refers to network slowdowns that occur because hardware or software has not been set up correctly. Poor cabling, misconfigured interface cards, or bad drivers can *seriously* reduce the performance of a network—by 90 percent or even more.

These problems can usually be corrected, but only if you are looking for them. Driver problems are particularly insidious because the natural tendency is for people to blame hardware when slowdowns occur. However, you cannot get the most of your hardware devices without proper software to run it. These issues are much more significant with bleeding-edge hardware than with established products, incidentally.

Also included in this category of issues are problems that occur due to poor design. For example, putting 30 busy workstations on a shared 10 Mbps Ethernet segment is likely to result in poor performance—using a switch would be much better. Another common mistake is not providing a “fatter pipe” (higher

bandwidth connection) to servers in a client/server setup. These issues can be avoided or ameliorated by reconfiguring the network—or even better, by designing it properly in the first place, right?

## **Asymmetry**

Bear in mind that many networking technologies, especially ones used for Internet access, are *asymmetric*, meaning that they offer much higher bandwidth in one direction than the other. Usually, this is arranged so that more bandwidth goes down to the user than from the user to the network, since most Internet users download far more than they upload. However, it's always important to find out if a speed rating is for both directions, or for only one direction, and if so, what the other direction's speed is. Common technologies with asymmetric performance include 56K modems, Asymmetric Digital Subscriber Line (ADSL), cable modems, and satellite Internet access. Beware, because the marketing people who sell these technologies will often try to hide the asymmetry of their services, usually highlighting only the bigger download figure and avoiding mention of the slower uploads.

Asymmetry can also have unexpected effects on network performance, because most communications, even if they seem unidirectional, are not. The most common case is when an Internet access technology has much higher download bandwidth than upload bandwidth. When using TCP/IP to download data, acknowledgments must be sent regularly. If the upstream bandwidth is too low, this may make it impossible to fully exploit the download bandwidth of the link.

**KEY CONCEPT** The theoretical rated speed of a network is never achieved in practice for a number of reasons. Overhead issues mean that not all of the possible capacity of a network can be used for data. External factors such as hardware bandwidth limitations restrict data input and output. Configuration problems can also greatly reduce real-world performance. Finally, it is important to remember that many technologies are asymmetric, offering higher speed in one direction than the other, and often, the larger number is the one that is advertised.

## **Simplex, Full-Duplex, and Half-Duplex Operation**

Another aspect of performance that is worthy of some attention is the mode of operation of the network or connection. Obviously, whenever we connect together Device A and Device B, there must be some way for Device A to send to Device B and Device B to send to Device A. Many people don't realize, however, that networking technologies can differ in terms of how these two directions of communication are handled. Depending on how the network is set up and the characteristics of the technologies used, you may be able to improve performance through the selection of performance-enhancing modes.

Let's begin with a look at the three basic modes of operation that can exist for any network connection, communications channel, or interface.

## **Simplex Operation**

In *simplex* operation, a network cable or communications channel can send information in only one direction; it's a one-way street. This may seem counter-intuitive: What's the point of communications that travel in only one direction? In fact, there are at least two different places in which simplex operation is encountered in modern networking.

The first is when two distinct channels are used for communication: one transmits from A to B and the other from B to A. This is surprisingly common, even though it isn't always obvious. For example, most, if not all, fiber-optic communication is simplex, meaning that it uses one strand to send data in each direction. But this may not be obvious if the pair of fiber strands are combined into one cable.

Simplex operation is also used in special types of technologies, especially ones that are asymmetric. For example, one type of satellite Internet access sends data over the satellite only for downloads, while a regular dial-up modem is used for upload to the service provider. In this case, both the satellite link and the dial-up connection are operating in a simplex mode.

## **Half-Duplex Operation**

Technologies that employ *half-duplex* operation are capable of sending information in both directions between two nodes, but only one direction or the other can be utilized at a time. This is a fairly common mode of operation when there is only a single network medium (cable, radio frequency, and so forth) between devices.

While this term is often used to describe the behavior of a pair of devices, it can refer more generally to any number of connected devices that take turns transmitting information. For example, in conventional Ethernet networks, any device can transmit, but only one may do so at a time. For this reason, regular (unswitched) Ethernet networks are often said to be half-duplex, even though it may seem strange to describe a LAN that way.

## **Full-Duplex Operation**

In *full-duplex* operation, a connection between two devices is capable of sending data in both directions simultaneously. Full-duplex channels can be constructed either as a pair of simplex links (as described earlier) or by using one channel that's designed to permit bidirectional simultaneous transmissions. A full-duplex link can connect only two devices, so many such links are required if multiple devices are to be connected together.

**NOTE** *The term full-duplex is somewhat redundant; duplex would suffice, but everyone still says full-duplex (likely, to differentiate this mode from half-duplex).*

Of these three options, full-duplex is obviously the one that yields the highest performance. Full-duplex operation doubles the theoretical bandwidth of the connection. If a link normally runs at 1 Mbps but can work in full-duplex mode, it really has 2 Mbps of bandwidth (1 Mbps in each direction). Remember the key word *theoretical*, however—you do not really get double the performance in real life,

because communications usually do not involve sending a lot of data in both directions at once. However, you certainly get better throughput than you do in a half-duplex mode.

In some cases, the mode of operation is a function of the technology and cannot be changed. In others, however, full-duplex mode is a matter of the correct hardware settings, and also whether the software supports full-duplex operation. Thus, getting higher performance in this area is sometimes simply a matter of ensuring proper configuration.

Full-duplex operation has been pretty much taken for granted in communications for years. The more interesting development has been the rise in the significance of full-duplex operation for local area networking. Traditionally, LANs have always used half-duplex operation on a shared access medium. As the use of switches has increased, thereby allowing dedicated bandwidth to each computer, full-duplex operation has become very popular. Full-duplex operation in Ethernet not only allows the simultaneous transmission of data in both directions, but also eliminates contention for the formerly shared access medium—thus, no more collisions. The combination of these two effects improves performance, sometimes substantially.

**KEY CONCEPT** There are three basic operating modes that describe how data is sent between connected devices on a network. In a *simplex* operation, data can flow in only one direction between two devices. *Half-duplex* networks allow any device to transmit, but only one may do so at a time. *Full-duplex* operation means two attached devices can each transmit and receive simultaneously. The latter offers the greatest potential performance, because forcing one device to wait for another before sending data does not decrease throughput.

## Quality of Service (QoS)

In my discussion of common network performance measurements earlier in this chapter, I mentioned that there are many different aspects to network performance. I also introduced the concept of *latency*, which measures how long it takes for data to travel across a network. Latency is one important part of a larger issue in networking that is sometimes called *quality of service* or *QoS*.

The inherent nature of most networking technologies is that they are more concerned with pumping data from one place to another as fast as possible than they are with how the data is sent. For example, the Internet is designed on top of the Internet Protocol (IP), a packet-switching technology (described in Chapter 1) that is designed to get packets from point A to point B in the most effective way, without requiring the user to have any knowledge about what route will be taken. In fact, some packets in the same data stream may be sent along different routes. Packets may be stored for a while before being forwarded to their destination, or even dropped and retransmitted.

For most applications, such as simple file or message transfers, this is perfectly fine. However, there are applications for which this sort of service represents low quality. In these cases, the nature of how the data is delivered is more important

than merely how fast it is, and there is a need for technologies or protocols that offer QoS. This general term can encompass a number of related features such as the following:

**Bandwidth Reservation** The ability to reserve a portion of bandwidth in a network or interface for a period of time so that two devices can count on having that bandwidth for a particular operation. This is used for multimedia applications for which data must be streamed in real time, and packet rerouting and retransmission would result in problems. This is also called *resource reservation*.

**Latency Management** A feature that limits the latency between two devices in any data transfer to a known value.

**Traffic Prioritization** In conventional networks, all packets are created equal. A useful QoS feature is the ability to handle packets so that more important connections receive priority over less important ones.

**Traffic Shaping** This refers to the use of buffers and limits, both of which restrict traffic across a connection to a value below a predetermined maximum.

**Network Congestion Avoidance** This QoS feature refers to monitoring particular connections in a network and rerouting data when a particular part of the network is becoming congested.

So, in essence, QoS in the networking context is analogous to QoS in the real world. It is the difference between getting take-out and sit-down service at a nice French restaurant—both cure the hunger pangs, but they meet very different needs. Some applications, especially multimedia applications such as voice, music, and video, are time dependent and require a constant flow of information more than raw bandwidth.

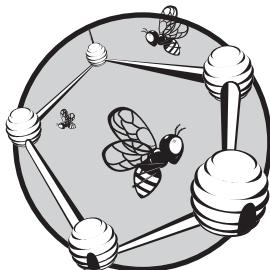
**KEY CONCEPT** The generic term *quality of service (QoS)* describes the characteristics of how data is transmitted between devices, rather than how quickly it is sent. QoS features seek to provide more predictable streams of data rather than faster ones. Examples of such features include bandwidth reservation, latency minimums, traffic prioritization and shaping, and congestion limitation. QoS is more important for specialty applications, such as multimedia, than for routine applications, such as those that transfer files or messages.

To support QoS requirements, many newer technologies have been developed or enhanced to add QoS features to them. This includes the ability to support *isochronous transmissions* that can reserve a specific amount of bandwidth over time to support applications that must send data in real time. One technology that has received a lot of attention for its QoS features is Asynchronous Transfer Mode (ATM). ATM is designed to support traffic-management features that are not generally available on networks that haven't been optimized to provide QoS features (such as Ethernet).

QoS has become a big buzzword lately. By itself, this term conveys about as much useful information about what the technology offers as being told that it is high performance. You have to dig past the marketing-speak and find out exactly what QoS features are being offered.

# 3

## NETWORK STANDARDS AND STANDARDS ORGANIZATIONS



You can't study networking and its related technologies without very quickly encountering a whole host of standards that are related to the subject, as well as the organizations that create these standards. Network standards facilitate the interoperability of network technologies and are extremely important. It may be an exaggeration to say that networking would not exist without standards, but it isn't an exaggeration to say that networking *as we know it* would not exist without them. Networks are literally everywhere, and every hardware device or protocol is governed by at least one standard and usually many.

In this chapter, I briefly examine the often overlooked subject of network standards and standards organizations. I begin with a background discussion of why standards are important, highlighting the differences between proprietary, de facto, and open standards. I give an overview of networking standards in general terms and then describe the most important international standards organizations and industry groups related to networking. I then describe the structure of the organizations responsible for

Internet standards, including the registration authorities and registries that manage resources such as addresses, domain names, and protocol values. I conclude with a discussion of the Request for Comment (RFC) process used for creating Internet standards.

## Proprietary, Open, and De Facto Standards

Why are standards important? An old saw in the computer world says, “The beauty of standards is that there are so many to choose from.” This little joke reflects the frustration that technicians often feel at the thousands of standards that are found in the industry. Aside from differing in terms of content—what technologies and protocols they describe—standards also often differ in terms of their type and how they came about. In fact, part of the reason why there are sometimes so many to choose from in a particular area is *because* of how they come about.

### ***Proprietary Standards***

In the early days of computing, many people didn’t quite understand just how important universal standards were. Most companies were run by skilled inventors, who came up with great ideas for new technologies, but who weren’t particularly interested in sharing them. It wasn’t considered a smart business move to share information about new inventions with other companies—the competition! Oh sure, companies believed that standards were important, but they thought it was even more important that *they* be the ones to control those standards.

For example, imagine that it’s 1985, and I have just come up with a great networking technology, which I have incorporated into a fancy new local area networking (LAN) product called SooperDooperNet. SooperDooperNet is *my* product. I have patents on the technology, I control its design and manufacture, and I sure as heck don’t tell anyone else how it works, because if I did, someone would copy me.

I could sell interface cards, cables, and accessories for SooperDooperNet, and companies that wanted to use it could install the cards in all of their PCs and be assured that those computers would be able to talk to each other. This solves the interoperability problem for this company by creating a “SooperDooperNet standard.” This would be an example of a *proprietary* standard—it’s owned by one company or person.

The problem with proprietary standards is that other companies are excluded from the standard development process, and therefore have little incentive to cooperate with the standard owner. In fact, just the opposite: They have a strong motivation to develop a competing proprietary standard, even if it doesn’t improve on the existing one.

So when my competition sees what I am doing, he is not going to also create network interface cards that can work with SooperDooperNet, which would require paying me a royalty. Instead, he is going to develop a new line of networking hardware called MegaAwesomeNet, which is very similar to SooperDooperNet in operation but uses different connectors, cable, and logic. He, too, will try to sell bunches of cards and cables to my customers.

The problem here is that the market ends up with different companies using different products that don't work together. If you install SooperDooperNet, you *have* to come to me for any upgrades or changes—you have no choice. Worse, what happens if Acme Manufacturing, which has 50 PCs running SooperDooperNet, merges with Emca Manufacturing, which has 40 PCs running MegaAwesomeNet? The IT people have a problem. Sure, there would be ways to solve it, but wouldn't everyone be better off avoiding these difficulties in the first place? And how could you create something like the Internet if everyone's networks use different "standards"?

## **Open Standards**

Eventually, companies learned that they would be better off with standards that everyone agreed on. This is particularly true of networking, where devices need to talk to each other. If many companies get together and agree to cooperate, they can create an *open standard* instead of a bunch of proprietary ones. The name is rather self-explanatory; rather than being the closely guarded secret of one organization, an open standard is available to anyone who is interested in using it.

One key to the success of an open standard is a steering organization to promote it. Usually, a neutral, nonprofit trade association or working group is established to develop the standard, and the various for-profit hardware and software companies join this group and support it financially. These groups also work with standards approval bodies like the International Telecommunication Union (ITU) and International Organization for Standardization (ISO) to gain acceptance for their standards. These and other standards organizations are described in the "International Networking Standards Organizations" section later in this chapter.

Of course, the companies aren't doing this just to be nice to their customers. In creating open standards, they split the market-share pie among them, but they make the pie grow much larger by attracting more customers. Customers like open standards more than proprietary ones, because those standards give them more choices and increase their ability to interact with other companies, troubleshoot problems, hire skilled workers, and expand in the future. As for the companies, they still compete with their specific offerings, so it's not like they all end up making the same products. For all of these reasons, open standards are now far more common than proprietary ones.

However, the process involved in creating these standards is often a difficult one. In some cases, the standards organization will draft the standard from the ground up, but in others, it may select one technology as the basis for the standard from several that are submitted in what is commonly called a "technology bake-off." Thus, many different companies may come to the table with different approaches, each of them vying for selection as the standard for use by the group. Politics can cause groups to get bogged down for years fighting over various options, or even to split into multiple groups. Good examples are what occurred in the conflict between supporters of 100VG-AnyLAN and Fast Ethernet, and the problems with standards politics that have plagued the world of powerline networking.

Furthermore, there are still some companies that believe strongly in proprietary standards, because they really want to control and direct the market. One of the most famous (infamous) in this regard is Sony, a company that makes excellent hardware but frequently refuses to accept established standards. For this reason, some people avoid Sony's products, even though they are good, because they want to stick to industry standards.

### ***De Facto Standards***

This brings me to the third type of standard that is often seen in the computer world: the *de facto standard*. “De facto” is Latin for “in fact.” A de facto standard is one that is used as a universal standard just because, over time, it has been widely used, and not because the standard was developed and approved by a standards committee.

A good example of a de facto standard is the AT command set used by modems. Virtually all modems use it, but this acceptance didn’t result from an industry group agreeing to adopt and deploy it. Rather, it was developed unilaterally by Hayes, the pioneering modem company, and then adopted by virtually every other modem maker until it became a standard.

One reason why proprietary standards are still sometimes seen is that some companies want to produce a standard that will become so universally used that it becomes the de facto standard, thus giving them a leadership position in that market. Again, in my estimation, Sony falls into this category—the developers often want to do things their way and create proprietary standards that they try to promote using their powerful market presence.

Sometimes this succeeds, but often it does not, resulting in a fragmented market of incompatible products. An excellent example is when Sony created a new format for a digital camera’s flash memory (the Memory Stick) rather than using the CompactFlash format used by other camera manufacturers. The end result was that not everyone used memory sticks as Sony had hoped, and there were now two incompatible standards that increased confusion and yielded no real benefit to the customer.

**KEY CONCEPT** Networking standards can be classified as *proprietary*, *open*, or *de facto*.

Proprietary standards are owned by one particular organization. If that organization has sufficient market clout and the industry lacks alternatives to its standard, it may be adopted by the whole industry, becoming a de facto standard. Usually, however, differing proprietary standards compete with each other, resulting in a fragmented market. In contrast, open standards are not owned by anyone—they are created by neutral organizations to ensure that compatible products can be designed and developed by many different companies. This makes life easier for the customer and also promotes the market as a whole.

## **Networking Standards**

All networking technologies have standards associated with them. These are usually highly technical documents, and they often presume that the reader has a fair bit of knowledge about networking. If you aren’t an expert, you will probably have some difficulty understanding networking standards.

In fact, many technologies have quite a number of standards associated with them. A networking technology may have more than one standard for any or all of the following reasons:

- The original standard has been revised or updated.
- The technology is sufficiently complex that it needs to be described in more than one document.
- The technology borrows from or builds on documents used in related technologies.
- More than one organization has been involved in developing the technology.

Standards documents created in the United States are usually developed in English, but are also routinely translated into other languages. European standards are often published simultaneously in English, French, German, and perhaps other languages as well.

Today, virtually all networking standards are open standards, administered by a standards organization or industry group. As I explained in the previous section, open standards are more popular than proprietary ones in the computer industry, and that's especially true when it comes to networking. In fact, the few technologies for which there is no universally accepted open standard have been losing ground to those with open standards, particularly in the areas of wireless LANs and home networking. This pretty much proves how important an open process really is.

**NOTE** You'll find discussions of various standards throughout this book. These can usually be found in an overview chapter introducing each technology type, though the discussions of more complex protocols include a section discussing relevant standards.

## International Networking Standards Organizations

The rise of open standards has been a great boon to customers of computer and networking products, as well as to the manufacturers that sell to them. In order to facilitate the development of open standards, however, we need organizations that will coordinate the creation and publishing of these documents. Generally, these are nonprofit organizations that specifically take a neutral stance regarding technologies and work for the betterment of the industry as a whole.

Here is a selective list of some of the standards organizations that you are likely to encounter when reading about networking and the Internet:

**International Organization for Standardization (ISO)** Probably the biggest standards organization in the world, the ISO is really a federation of standards organizations from dozens of nations. In the networking world, the ISO is best known for its OSI Reference Model, which is discussed in Part I-2 of this book.

**NOTE** The shortened name of the International Organization for Standardization is indeed ISO, not IOS, as you might imagine. In fact, it is not an acronym at all. Since the full name of the body differs from one language to the next, any acronym for that name would differ as well. Instead, the organization chose the name ISO from the Greek word *isos*, meaning equal. Many people, especially in the United States, think ISO is short for International Standards Organization, but this is incorrect.

**American National Standards Institute (ANSI)** ANSI is the main organization responsible for coordinating and publishing computer and information technology standards in the United States. Although many people think that this organization develops and maintains standards, it does neither. Instead, it oversees and accredits the organizations that actually create the standards, qualifying them as *Standards Developing Organizations* or *SDOs*. ANSI also publishes the standards documents created by the SDOs and serves as the United States' representative to the ISO.

**Information Technology Industry Council (ITIC)** ITIC is a group of several dozen companies in the information technology (computer) industry. ITIC is the SDO approved by ANSI to develop and process standards related to many computer-related topics. It was formerly known as the *Computer and Business Equipment Manufacturers Association (CBEMA)*.

**National Committee for Information Technology (NCITS)** NCITS is a committee established by the ITIC to develop and maintain standards related to the information-technology world. NCITS was formerly known by the name *Accredited Standards Committee X3, Information Technology*, or more commonly, just *X3*. It maintains several subcommittees that develop and maintain standards for various technical subjects.

**Institute of Electrical and Electronics Engineers (IEEE)** The IEEE (pronounced “eye-triple-ee”) is a well-known professional organization for those in the electrical or electronics fields, including computers and networking. IEEE’s main claim to fame in the networking industry is the IEEE 802 Project, which encompasses many popular networking technologies, including Ethernet.

**Electronic Industries Alliance (EIA)** The EIA is an international industry association that is best known for publishing electrical wiring and transmission standards.

**Telecommunications Industry Association (TIA)** The TIA is the communications sector of the EIA, and it is responsible for developing communications standards. Since communications, wiring, and transmission are all related, and since the TIA and EIA organizations are also related, standards produced by the EIA or TIA are often labeled with the combined prefixes EIA/TIA or TIA/EIA.

**International Telecommunication Union—Telecommunication Standardization Sector (ITU-T)** ITU-T is another large international body that develops standards for the telecommunications industry. The ITU-T was formerly named the International Telephone and Telegraph Consultative Committee (CCITT; the abbreviation comes from the French version of the organization’s name: *Comité Consultatif International Téléphonique et Télégraphique*).

**European Telecommunications Standards Institute (ETSI)** An organization with members from dozens of countries both within and outside Europe that is dedicated to developing telecommunications standards for the European market (and elsewhere). ETSI is known for, among other things, regulating the use of radio bandwidth in Europe and developing standards such as HiperLAN.

Many of these organizations do not actually develop the various standards. Generally, these are oversight organizations—high-level management, if you will—that work with many other smaller groups who actually develop the standards. Also, in many cases, a particular standard may be published by more than one standards organization, so it may be labeled with more than one name.

**NOTE** *The set of related organizations responsible for creating Internet standards is not shown in this list because I have elected to cover them in two dedicated sections later in this chapter, on Internet standards organizations and registration authorities.*

**KEY CONCEPT** There are a number of well-known international organizations that play important roles in the development of open networking standards. Some of the most important of these are ISO, ANSI, ITIC, IEEE, EIA/TIA, ITU-T, and ETSI. These are oversight organizations, responsible for overall management of the standards development process, rather than for the particulars of creating individual standards.

## Networking Industry Groups

While most open standards are coordinated and published by a small number of large, often international, standards organizations, these are not the only groups involved in the development of standards for networking and Internet technologies. Many different networking *industry groups* play an important role in the standard creation process.

Networking industry groups differ in a few ways from standards organizations. They are typically dedicated to the promotion of a specific technology, whereas standards organizations are more generic and oversee hundreds of different ones. Industry groups are also generally smaller than standards organizations, with members drawn primarily from the field of developers and manufacturers that create products for the particular technology the group promotes.

Perhaps most important, industry groups often actually write and maintain the standards, whereas standards organizations generally act as supervisors who ensure that the standards are clear enough to be implemented. Some industry groups, however, are concerned only with marketing and promotion activities.

Obviously, these industry groups work closely together with the standards organizations. In some cases, they may even be part of the same overall organization, and all of the different groups are related in some way. For example, the IEEE 802 Project consists of a number of working groups charged with maintaining and developing specific technology standards, which the larger IEEE organization approves and publishes.

One of these working groups is the 802.11 working group, which develops wireless Ethernet technology. At the same time that this group does its thing, there is an industry group called the *Wireless Ethernet Compatibility Alliance (WECA)*. This group works to ensure the cross-vendor compatibility of 802.11b wireless networking hardware and software.

Other industry groups are formed specifically to develop independent standards that are not approved through a formal standardization process. Examples include groups such as HomePNA, IrDA, and HomeRF.

One of the problems with these groups is that they usually do not make their standards open to the public. This is undoubtedly due to some sort of security concern or desire to keep the inner workings of their technology secret. Unfortunately for these groups, this policy harms the ability of regular people to learn how their technologies work.

**NOTE** *As an example of what I mean about these closed standards, I can point to my own experience in writing this and other reference works. I was almost always unable to obtain specifications from most of the private industry groups. They either refused to allow me to get the document at all or wanted to charge me a great deal of money for the privilege (well into the thousands of dollars in some cases). In doing this, these groups harm their own cause, thereby making it more difficult for those interested in their technologies to learn about them. This is another key advantage of having open standards managed by public organizations such as ANSI or the IEEE.*

## Internet Standards Organizations (ISOC, IAB, IESG, IETF, IRSG, and IRTF)

High-quality, widely accepted open standards become more important as the number of people that use a network grows. The largest network of all is of course the *Internet*, which connects millions of people on thousands of individual networks into a globe-spanning internetwork. The Internet has revolutionized not only networking and computing, but also communication, business, and even society as a whole. One of the critical factors in the success of the Internet has been its development using open standards.

Of course, nobody sat down one day and said, “Hey, let’s create the Internet!” (No, not even Al Gore.) It began as a small research network, and was developed over time concurrently with the technology set that implemented it: TCP/IP. At first, a relatively small organization was sufficient for managing the development of Internet standards and overseeing its activities, but as the Internet continued to grow, this organization became inadequate. Eventually, a more formalized structure of organizations was required in order to manage the Internet development process and other activities. This ensured the continued success and growth of the Internet and the TCP/IP technologies that powered it.

Today, six organizations are responsible for the development of the Internet’s architecture, standards and policies, and related activities. They are closely related, with certain organizations responsible for overseeing others. These organizations perform many tasks and can be somewhat confusing to understand, since many have similar-sounding names and responsibilities. Therefore, I will concentrate mostly on their role in the development of Internet standards, since that is the primary interest in this discussion.

Here are brief descriptions, rather simplified, of the key Internet standards organizations:

**Internet Society (ISOC)** A professional society responsible for general, high-level activities related to the management, development, and promotion of the Internet. ISOC has thousands of individual and organizational members that engage in activities such as research, education, public policy development, and standardization.

It is responsible for providing financial and administrative support to the other organizations listed in this chapter. From the standpoint of standards development, ISOC's key role is its responsibility for oversight of the IAB.

**Internet Architecture Board (IAB)** Formerly the *Internet Activities Board*, the IAB is charged with the overall management of the development of Internet standards. It makes “big-picture” policy decisions related to how Internet technologies and structures should work. This ensures that various standardization efforts are coordinated and consistent with overall development of the Internet. It is responsible for publishing Internet standards (RFCs), as described in the “Internet Standards and the Request for Comment (RFC) Process” section at the end of this chapter. It advises the ISOC and oversees the IETF and IRTF; it also acts as an appeals body for complaints about the standardization activities performed by the IETF. The charter of the IAB is described in RFC 2850.

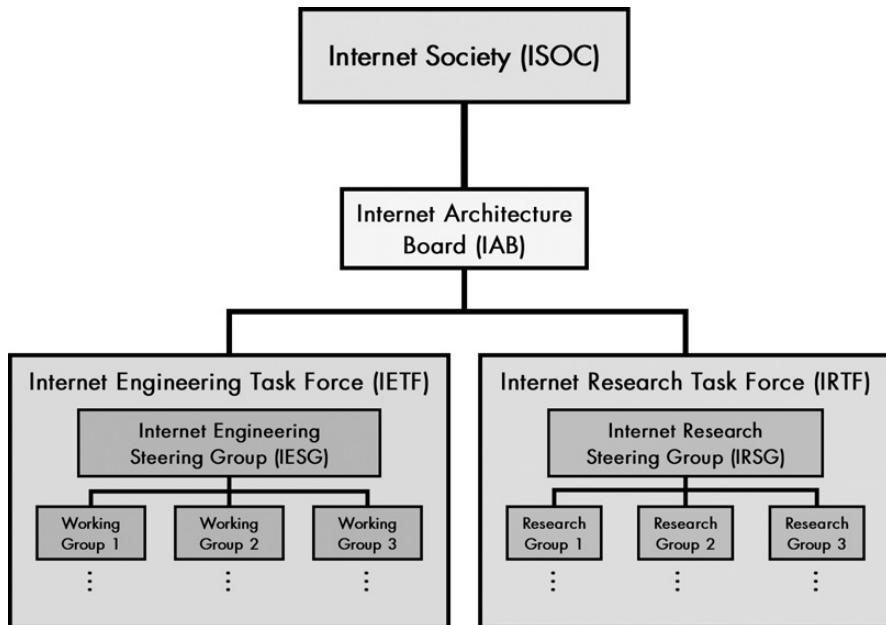
**Internet Engineering Task Force (IETF)** The IETF focuses on issues related to the development of current Internet and TCP/IP technologies. It is divided into a number of *working groups*, each of which is responsible for developing standards and technologies in a particular area, such as routing or security. Each area is managed by an *area director*, who serves on the IESG. The IETF is overseen directly by the IESG and in turn by the IAB; it is described in RFC 3160.

**Internet Engineering Steering Group (IESG)** The IESG is directly responsible for managing the IETF and the Internet standards development process. It consists of each of the IETF area directors, who make final decisions about the approval of proposed standards, and works to resolve any issues that may arise in the standardization process. The IESG is technically considered part of the IETF and is also described in RFC 3160.

**Internet Research Task Force (IRTF)** Where the IETF is focused primarily on short-term development issues, the IRTF is responsible for longer-term research related to the Internet and TCP/IP technologies. It is a much smaller organization than the IETF, consisting of a set of *research groups*, which are analogous to the IETF’s working groups. The IRTF is overseen by the IRSG and IAB. It is described in RFC 2014.

**Internet Research Steering Group (IRSG)** The IRSG manages the IRTF in a similar way to how the IESG manages the IETF. It consists of the chairs of each of the IRTF research groups and works with the chair of the whole IRTF to make appropriate decisions on research activities. It is also discussed in RFC 2014.

Figure 3-1 shows the relationship between the Internet standards associations. The ISOC oversees the IAB, which in turn directs the IETF and IRTF. The IETF develops current Internet and TCP/IP standards and is headed by the IESG, which manages IETF working groups. The IRTF is the IETF’s research counterpart, containing *research groups* led by the IRSG.



**Figure 3-1: Internet standards organizations** The ISOC is responsible for overseeing the IAB, which in turn is responsible for the two task forces, the IETF and IRTF, which are headed by the IESG and IRSG, respectively.

Of these organizations, the IETF is the one that you will most often hear referenced, because it is directly responsible for the development of the majority of Internet standards. Thus, whenever I mention Internet standards development efforts in this book, I'm referring to the IETF as the organization doing the work. This is, of course, a bit of an oversimplification, since all of these organizations play a role in the standards development process, as described later in this chapter, in the discussion of the RFC process.

Many of these organizations are responsible for a great deal more than just standards development. This is especially true of the ISOC, for which standardization is just one of many activities. The IAB also performs a number of functions not strictly associated with standards development, including managing the assignment of protocol values done by the Internet Assigned Numbers Authority and acting as a liaison between the Internet standards organizations and other standards bodies.

**KEY CONCEPT** A group of related organizations is responsible for the development of TCP/IP standards and Internet technologies. The *Internet Society (ISOC)* has overall responsibility for many Internet activities, including standards development. It oversees the *Internet Architecture Board (IAB)*, which makes high-level decisions about Internet technology development. Most of the actual work of creating current Internet standards is performed by the *Internet Engineering Task Force (IETF)*, which is managed by the *Internet Engineering Steering Group (IESG)*. Longer-term research is done by the IETF's sibling organization, the *Internet Research Task Force (IRTF)*, which is led by the *Internet Research Steering Group (IRSG)*.

## **Internet Registration Authorities and Registries (IANA, ICANN, APNIC, ARIN, LACNIC, and RIPE NCC)**

The success of the global Internet relies on the development of universally accepted standards for protocols and other technologies. Internet standards organizations such as the IETF are thus critically important. They manage the standards development process, which ensures that everyone agrees on how to create hardware and software that will work together to communicate worldwide.

While the need to standardize protocols seems obvious, there are a couple of other aspects to Internet standardization that are equally important but perhaps not quite as well understood:

**Parameter Standardization** Most protocols rely on the use of parameters that control how they function. As just two of many, many examples, the IP has a set of numbers that define different IP options, and the Address Resolution Protocol (ARP) has an Operation Code field that can take on many different values. Just as it is essential for devices to agree on what protocols to use, so they must also agree on what parameters to use for those protocols, if communication is to be successful.

**Global Resource Allocation and Identifier Uniqueness** There are a number of resources that are used on the Internet that must be allocated from a fixed set of values. Uniqueness in assignment is essential for these values. The most obvious example is that each TCP/IP host must have a unique IP address. Another important example is ensuring that only one organization uses a given Domain Name System (DNS) domain name. If two devices have the same IP address or two organizations try to use the same domain name, the results would be unpredictable, but almost certainly bad!

In both of these cases, some sort of centralized organization is required. We need a group to take responsibility for managing parameters. It must ensure that everyone uses the same parameters, and the same protocols. We also need to coordinate the assignment of identifiers such as addresses and names. This ensures that the identifiers are created and allocated in a way that is acceptable to all. In the world of the Internet, these are sometimes called *management authorities* or *registration authorities*.

### ***Internet Centralized Registration Authorities***

The organization originally responsible for managing parameters and identifiers was the Internet Assigned Numbers Authority (IANA). Amazingly, while the name makes it sound like the IANA was a huge bureaucracy, it was effectively one man: Jonathan B. (Jon) Postel, one of the most important pioneers of Internet and TCP/IP technologies. Jon Postel ran IANA until his untimely death in 1998.

IANA was originally charged with managing which IP address blocks had been assigned to different companies and groups, and it maintained lists of periodically published Internet parameters such as UDP and TCP port numbers. It also was in charge of the registrations of DNS domain names, which were more directly handled

by the Internet Network Information Center (InterNIC), a service managed by the United States government. Network Solutions, Inc. (NSI) was later granted the contract to manage the InterNIC and was eventually purchased by VeriSign.

As the Internet continued to grow, an effort commenced in the mid-1990s to define a new organization that would be responsible for the central registration of Internet addresses and names. This took the form of a new private, nonprofit company called the Internet Corporation for Assigned Names and Numbers (ICANN). ICANN is officially charged with all of the centralized registration tasks I have mentioned so far, including IP address assignment, DNS domain name assignment, and protocol parameters management.

In a simpler world, this development would have meant that ICANN would have replaced IANA, which would no longer exist. Instead, ICANN kept IANA around, leaving that organization in charge of overseeing IP address registration and Internet parameters. ICANN is now in charge of IANA, so both organizations are responsible for IP addresses and parameters. This often leads to confusion, and to make things worse, it is common to see IANA and ICANN mentioned in conjunction as IANA/ICANN or ICANN/IANA.

**KEY CONCEPT** Internet registration authorities are centralized organizations responsible for coordinating protocol parameters and globally assigned resources such as IP addresses. The first such organization was the *Internet Assigned Numbers Authority (IANA)*, which was initially in charge of IP address assignment, DNS domain name management, and protocol parameters. Today, the *Internet Corporation for Assigned Names and Numbers (ICANN)* has overall responsibility for these activities; the IANA operates under the auspices of ICANN and is still responsible for IP address assignment and parameter coordination.

## ***Modern Hierarchy of Registration Authorities***

In the original “classful” IP addressing scheme, addresses were assigned to organizations directly by IANA in address blocks: Class A, Class B, and Class C. Today, a hierarchical, classless addressing system called *Classless Inter-Domain Routing (CIDR)* is used instead. Address assignment in CIDR involves the hierarchical allocation of blocks of addresses, starting with large blocks that are given to big organizations, which split them to assign to smaller groups. (Much more detail on these methods can be found in Chapters 16 through 20, which cover IP addressing.)

IANA, as the organization in charge of all IP addresses, assigns the largest blocks of addresses to *regional Internet registries (RIRs)* that are responsible for further allocation activities. Each RIR manages IP addresses and other Internet number resources (such as autonomous system numbers) for a particular region. The four regional registries are as follows:

**Asia Pacific Network Information Centre (APNIC)** Covers the Asia/Pacific region.

**American Registry for Internet Numbers (ARIN)** Manages North America, part of the Caribbean, and subequatorial Africa.

**Latin American and Caribbean Internet Addresses Registry (LACNIC)** Responsible for Latin America and part of the Caribbean.

**Réseaux IP Européens Network Coordination Center (RIPE NCC)** Takes care of Europe, the Middle East, Central Asia, and Africa north of the equator.

Each registry may assign address blocks to Internet service providers (ISPs) directly or further delegate them to *national Internet registries* or smaller *local Internet registries*. (See Chapter 16, which covers IP address allocation issues, for more details.)

Name registration has changed over the last several years. It is no longer part of IANA's responsibilities, and ICANN has opened up the name registration business, so it is no longer the province of a single organization such as InterNIC/NSI/VeriSign. Now, many different accredited registrars can be used for name registration in many of the popular top-level domains. This is discussed in Chapter 54, which covers DNS public registration. The complete list of documents containing Internet and TCP/IP parameters can be found on the IANA's website at <http://www.iana.org/numbers.html>.

## Internet Standards and the Request for Comment (RFC) Process

The precursors of the modern Internet were diminutive networks developed and run by a small group of computer scientists and engineers. These technologists knew that developing open, widely adopted standards would be essential to the eventual growth of the Internet and the TCP/IP protocol suite. But there was no formalized standards development mechanism back then.

Standardization was achieved largely through building consensus through discussion about new technologies and protocols. If someone had a proposal for a new protocol or technology, or an idea for a change to an existing one, that person would create a memorandum describing it and circulate it to others. Since the goal was to solicit comments on the proposal, these memos were called *Requests for Comments (RFCs)*. Not all RFCs described formalized standards; many were just descriptive documents, clarifications, or contained miscellaneous information.

**NOTE** The documents defining early standards were originally called Internet Engineering Notes (IENs) before they were called RFCs.

Today, of course, the Internet is enormous, and there is an official structure of Internet standards organizations that is responsible for creating new Internet and TCP/IP standards. Due to the many thousands of people who play an active role in developing Internet technologies, an informal system where anyone could just write an RFC would lead to chaos. Thus, Internet and TCP/IP standards are still called RFCs, but the process of creating one is much more formal and organized today.

The IETF is the standards body that is most directly responsible for the creation of Internet standards. The IETF's working groups, overseen by the IESG and the IAB, develop new protocols and technologies continuously, and these developments are formalized in RFCs.

The office of the RFC Editor handles the publishing of RFCs. For nearly 30 years, beginning in 1969, the RFC Editor was Internet pioneer Jon Postel. After his death in 1998, the function was assigned to the networking division of the USC Information Sciences Institute (ISI), where Jon Postel was once director. The

function of the RFC Editor is to publish and archive RFCs, and to maintain an online repository of these documents so that they can be accessed and used by the Internet community.

The open and free access to RFCs has greatly contributed to the Internet's success. Even today, if you consider that standards bodies charge thousands of dollars for access to a single standard, the ability to log on and immediately retrieve any of the thousands of RFCs is noteworthy.

**NOTE** An up-to-date list of RFCs with hyperlinks to each document (except for some of the early ones) can be found at the office of the RFC Editor. Go to <http://www.rfc-editor.org/rfc-index.html>.

## RFC Categories

As I mentioned, not all RFCs are official Internet standards. This is important to remember. Each RFC has a *category* or *status* associated with it that indicates its disposition:

**Proposed Standard/Draft Standard/Standard** These documents describe technologies that are on the standards track. That means they are either already formally approved as standards, or they are likely to become standards in the future. In many cases, the document is just listed as “standards track,” rather than one of those three precise labels.

**Best Current Practice** A document providing guideline information or recommendations from the IETF that is not a formal standard.

**Informational** A document that provides general information or commentary.

**Experimental** A proposal for an experimental standard that is not on the standards track. In some cases, protocols or proposed changes to existing protocols that are not accepted as formal standards are changed to experimental status.

**Historic** Former standards that have been made obsolete.

## The Internet Standardization Process

Before a proposal will be considered for the Internet standardization process, it must be published as an *Internet Draft (ID)*. The IETF publishes a set of guidelines that specify how IDs must be created and submitted. Members of working groups within the IETF who are involved in specific projects write most IDs. However, because the standards process is open, any member of the public can independently submit a standard for review by creating an ID for consideration by the IETF and IESG. IDs are usually revised many times based on feedback from other working groups within the IETF.

If an ID has been reviewed and is considered valuable, well understood, and stable (meaning that it is not being rapidly updated with new revisions), it may become a candidate for standardization. The IESG can place the ID on the Internet standards track by changing its status to *proposed standard*. Documents of this status are considered mostly complete, but may still be revised based on further review, testing, and experimentation with the technology.

Once the specification is sufficiently mature and widely accepted, it may be elevated from proposed standard to *draft standard*. A key requirement for such advancement is that the technology must be demonstrated to be functional on at least two independent and interoperable implementations. This proves that the standard has been cleared and completed, and that at least two different groups have been able to implement it compatibly.

A document only reaches draft standard when the IETF community believes it is technically mature and the specification is complete. Changes are usually only made to draft standards to correct problems encountered in testing or resolve new issues that arise.

The final station on the Internet standards track is *Internet standard*. This designation is applied to only very mature specifications that are popular and that have been widely implemented. A document that reaches this status often describes a technology that is or will become universally implemented, and is assigned an STD (standard) number.

The RFC development process can take months or even years, depending on how complex the technology is, how many changes are required to the documents, and whether or not the proposal is considered important or interesting. Many RFCs never make it officially to Internet standard status; draft standard status is generally considered stable enough that the technology is often just implemented by companies when that level is reached. Some RFCs never even make it to draft standard status, and the technologies they describe are still used in products.

Once an RFC is published, it cannot be changed. This is a specific policy decision intended to avoid the confusion that would otherwise result from the fact that there were multiple versions of the same RFC. The RFC publication process incorporates a number of steps at which RFC authors can revise their documents and check for editorial omissions and errors.

This need for a new document whenever a change is made is also why proposals are typically published with a category designation of standards track rather than proposed standard, draft standard, and Internet standard. This eliminates the need to publish a new RFC when a proposal advances down the standards track without requiring any real changes aside from a different category designation.

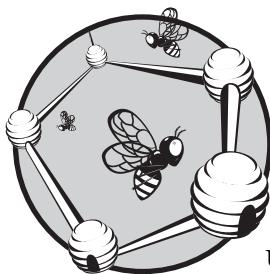
I've just outlined the process for creating and publishing an Internet standard here. The full details of the standards process can be found in RFC 2026 (where else but an RFC?).

**KEY CONCEPT** Internet standards are described in a series of documents called *Requests for Comments (RFCs)*. The RFC process describes how an Internet standard is usually created. An idea for a new technology or enhancement begins with the creation of an *Internet Draft (ID)*. After review and feedback, if the proposal has support, it may be placed on the Internet standards track, and its status will be changed to *proposed standard*. As the fledgling standard matures, its status may advance to *draft standard* and eventually, *Internet standard*. However, many RFCs are implemented in products without reaching Internet standard status. There are also other RFCs that define experimental technologies or provide information without describing official Internet standards.



# 4

## A REVIEW OF DATA REPRESENTATION AND THE MATHEMATICS OF COMPUTING



We use decimal (base 10) numbers to represent numeric information, and we use various alphabets and symbol systems to represent other types of information. In contrast, computers understand only one basic type of information: ones and zeros, which themselves are representative of either an on or off electrical state within the hardware of the device. These ones and zeros are combined in various ways to form more common data elements that we are used to finding in computers: regular numbers, characters, and files. However, all of these are really only abstractions; the ones and zeros are always underneath whatever logical structures are used within the computer.

This same basic foundation of ones and zeros applies to networking as well. Even though most of the information in a network is exchanged in a logical fashion between higher-layer protocols, ones and zeros sent over the network medium underlie all networking structures. Understanding how data is represented and manipulated in computer systems is important because it will help you comprehend many of the different technologies.

Computer data representation and mathematics are important for explaining how low-level physical layer modulation and encoding techniques work. Those two elements come into play even for higher-level concepts, such as how IP addresses are set up and used on the Internet.

In this chapter, I provide some general background information on how numerical data is represented, stored, and manipulated within computers and networking hardware. I begin with a description of binary numbers and the different terms used to refer to collections of binary information of various sizes. I describe the different types of numbering systems used in computer systems, such as octal, decimal, and hexadecimal, and how data can be converted between these different types. I explain how arithmetic is performed on binary and hexadecimal numbers. I then discuss boolean logic and how logical functions are used to manipulate binary data.

These explanations then form the basis for a discussion of how logical functions are used for setting, clearing, inverting, and masking bits. These operations are employed extensively in certain networking technologies and protocols. Masking operations especially are often used in IP addressing, so even though this section seems rather low-level, it is quite relevant to the world of TCP/IP.

**NOTE** *Needless to say, you may know most or all of the information in this chapter, so feel free to skip (or just skim) those topics that you already know. I provide this background detail for the sake of those new to computing or those needing a refresher. However, even those of you who know what a bit and a byte are, and know the difference between binary and decimal numbers, may find the discussion of bit masking worth perusing.*

## **Binary Information and Representation: Bits, Bytes, Nibbles, Octets, and Characters**

The essence of computing is *information*. Computer hardware and software are designed to allow the input, storage, transfer, and expression of various types of information. One primary way by which types of information are differentiated is as either *analog* or *digital*.

Consider, for example, a light switch and a dimmer. A light switch allows a light to be turned on or off; there are no in-between states. These discrete states, on or off, represent digital information. In contrast, a dimmer allows you to fine-tune the light output from fully on to fully off, with an infinite number of intermediate states in between; that's analog information.

### ***Binary Information***

Modern digital computers store information digitally. In the same way a light bulb has only an on or off value, so do the components that store and manipulate information within computers. Millions of *transistors* compose computer processors and other circuits, and are, in highly simplified form, digital switches. Thus, all information in computers is manipulated as collections of information pieces that can be only on or off, like a switch.

Since there are only two possible states—on or off—this is called *binary* information (the prefix *bi* means two). There are several advantages to using binary representation for information. It is a simple way to represent many types of information, whether a light switch is on or off or a file has been successfully copied. It is also possible to combine binary values to represent more complex information.

Perhaps most important, binary information is *unambiguous*: On is always on, and off is always off. This property is important because it allows devices to detect clearly the value of a particular piece of information. Computers like black and white; they are not particularly good at dealing with shades of gray. (This becomes especially important in the field of networking, in which transmission of data can cause signals to become polluted by noise.)

The on or off condition of a binary value can be expressed in a number of different ways. In logical expressions, we may consider the value to be true or false. When representing mathematical values, the most common representation is one (on) or zero (off).

### ***Binary Information Representation and Groups***

The fundamental building block of computer information is the *bit* (a contraction of *binary digit*). Every bit can be either 0 or 1. Making the value of a bit 1 is commonly called *setting* the bit; changing it to 0 is *resetting* or *clearing* it.

Of course, bits represent only a very small amount of information: a single fact or value. We must make collections of these bits so that we can use them to store large amounts of information and more complex data types. The most common grouping is to take 8 bits and reference them as a single unit. A collection of 8 bits is technically called an *octet*, but is more commonly called a *byte* (more on that in a moment).

*Byte* is a jocular play on the term *bit*. Over time, various sizes of bit collections have been defined. Some geek comedian decided that if 8 bits made a byte, then 4 bits must be a *nybble* (or “nibble”). Hilarious, no? Larger collections have also been defined and given various names. Table 4-1 summarizes the most common representations of groups of bits and the terms used for them; their relative sizes are also shown graphically in Figure 4-1.

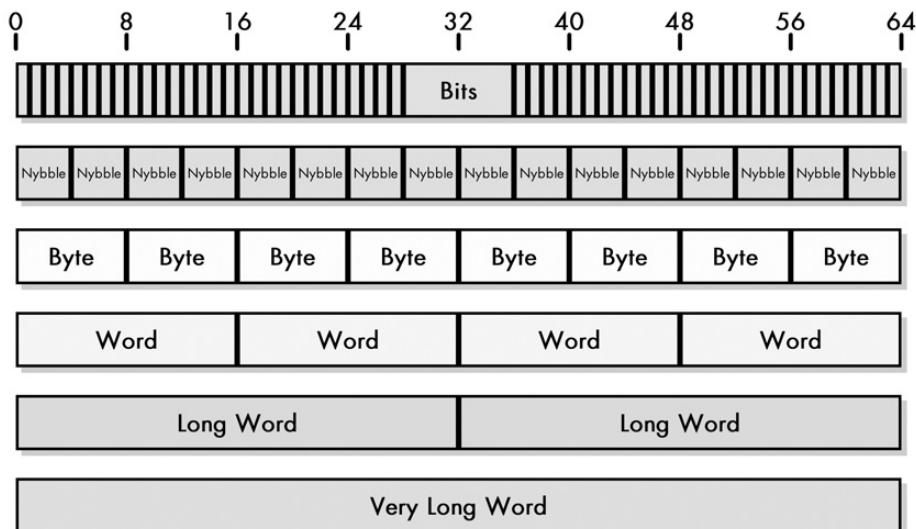
**Table 4-1:** Binary Information Group Representations and Terms

<b>Number of Bits</b>	<b>Common Representation Terms</b>
1	Bit/Digit/Flag
4	Nybble/Nibble
8	Byte/Octet/Character
16	Double Byte/Word
32	Double Word/Long Word
64	Very Long Word

A few of the new terms that appear in Table 4-1 are worth special mention. A bit is also sometimes called a *flag*; this term is most often heard when a bit is used by itself to represent a particular information state. For example, a computer might

use a *Changed* flag to represent whether a particular file has been modified; this is an analogy to a flag either being raised or lowered to indicate a condition. These flags are often seen in networking message formats.

The term *character* is also used to express a set of 8 bits. This use comes from the fact that computers often store alphanumeric characters, such as letters and numbers, one to a byte. The 16-bit *word* is used fairly often, but not nearly as much as *byte*. The larger collections of bits, such as double word and so on, are not often encountered in everyday parlance; they are used to represent chunks of data in technical fields such as hardware design or programming.



**Figure 4-1: Binary information representations and terms** This diagram shows the relative sizes of the most commonly sized collections of binary information.

Notice that the number of bits used for each of these terms is a power of two. As you will see later in this section, this occurs because when bits come in sets that are a power of two in size, they are easier to represent and manipulate.

**KEY CONCEPT** Computers store all information in *binary digital form*. This means that all data—whether it's text, photographs, audio, or whatever else—is composed of only collections of ones and zeros. The fundamental building block of digital information is the *binary digit* or *bit*, which represents a single zero or one state. To represent larger amounts of information, bits can be collected into groups of 4, 8, 16, 32, or 64, called *nybbles*, *bytes*, *words*, *long words*, and *very long words*, respectively.

## Byte Versus Octet

There has been some disagreement, and even controversy, surrounding the use of the words *byte* and *octet*. The word *byte* has traditionally been the most commonly used term for a set of 8 bits, especially in North America. However, it is *technically* not the correct term.

A byte is, formally, the smallest unit of data that can be read from or written to at one time in a computer system. In almost all cases today, that is indeed 8 bits, but there have been some systems in which a byte was not 8 bits. Some older 36-bit computers used 9-bit bytes, and others had byte sizes of 6 or 7 bits, or even variable-sized bytes. For this reason, many people, especially techie professionals, prefer the term *octet*, which clearly and unambiguously implies 8. This term is much more common outside North America.

**NOTE** *This matter of octets and bytes is the kind of tempest in a teapot that computer people love so much. The bottom line in modern computer systems, however, is that an octet is a byte and a byte is an octet, and the terms can generally be used interchangeably without too much danger. You will more often see octets used in technical standards. In this book, I use the term bytes because it is the term that most people are familiar with.*

**KEY CONCEPT** Formally, an *octet* is the correct term for exactly 8 bits, while a *byte* is the smallest number of bits that can be accessed in a computer system, which may or may not equal 8. In practice, modern computers use 8-bit bytes, and the terms are used interchangeably (with *byte* being more common in North America, and *octet* often being preferred in Europe).

## Decimal, Binary, Octal, and Hexadecimal Numbers

The numbers we are accustomed to using in everyday life are called *decimal numbers*. The word *decimal* refers to the number 10. Every digit can take on one of ten values: 0 to 9. Arithmetic performed on decimal numbers is also called *base 10* mathematics, because of this orientation around the number 10. (Why is the number 10 the foundation of our normal mathematical system? Hold both hands up and count!)

Computer systems, however, don't have fingers or toes; they deal only with binary numbers, which have just two values. Each bit can represent only a 0 or a 1. A single 0 or 1 value is sufficient for encoding a single fact, such as whether something is true or false, or whether the answer is yes or no. But a bit is not enough to hold more complex information, such as your bank account balance, a text document, or a picture of the Yellowstone Canyon.

### ***Binary Numbers and Their Decimal Equivalents***

For this reason, larger collections of bits have been created by computer scientists, such as bytes (octets), words, and so forth. When individual bits are collected into sets in this way, they can represent larger integers, called *binary numbers*. Since there are only two possible values for each digit in a binary number (0 or 1), binary numbers are also called *base 2* numbers.

The key to understanding binary numbers is to realize that they are exactly the same as decimal numbers, except that each digit has a value in the range of 0 to 1, instead of 0 to 9. For example, when you count in decimals, you go up to 9 in the ones place, and then you need a second place for tens. If you go above 99, you need a third place for hundreds. Each additional place added on the left is a higher power of ten.

Binary is the same, except the limit for each place is 1 instead of 9. So, in binary, you go up to 1 in the ones place, and then need a second place for twos (instead of tens). If you go above 3, you need a third place for fours (instead of hundreds). Each added digit is a subsequent higher power of two, rather than ten.

Thus, where counting in decimal goes 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, and so on, counting in binary goes 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101. For example, the number 13 in decimal is the same as 1101 in binary. How? Well, in decimal, we have a 3 in the ones place, plus a 1 in the tens place, which has a value of 10. This is  $3 + 10$ , or 13. In binary, we start with a 1 in the ones place, add a 1 in the fours place (for a value of 4), plus a 1 in the eights place, for a value of 8. This is  $1 + 4 + 8$ , or 13.

To take a more complex example, 211 in decimal is 11010011 in binary. Table 4-2 shows how the two are equivalent, by adding the values for each binary digit place where there is a 1. Read it from left to right, going top to bottom. Starting in the leftmost column, you can see that the example number has a 1 in the 128s place. So you start with a sum of 128. In the next column there is a 1 in the 64s place, so you add 64 for a running sum of 192. But in the 32s place, the binary digit value is 0, so you don't add 32 to the sum. If you continue down to the ones place, you'll get the decimal equivalent of the binary number.

**Table 4-2:** Binary and Decimal Number Equivalents

Binary Number	1	1	0	1	0	0	1	1
Power of Two	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Value of Digit Place	128	64	32	16	8	4	2	1
Value for This Number	128	64	0	16	0	0	2	1
Running Sum (from Left to Right)	128	$128+64 = 192$	192	$192+16 = 208$	208	208	$208+2 = 210$	$210+1 = \mathbf{211}$

As you can see, a binary number with  $N$  digits can hold up to  $2^N$  values. So a byte with 8 bits can hold  $2^8$ , or 256 different values, which are numbered from 0 to 255. A 16-bit word can hold  $2^{16}$ , or 65,536 values.

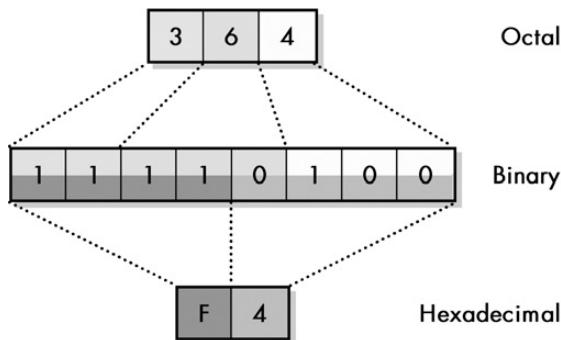
### **Making Binary Numbers Easier to Use by Grouping Bits**

One problem with binary numbers is that although computers love them, people have trouble with them because they quickly become long and cumbersome to deal with. For example, 1,000,000 in the decimal system is 11110100001001000000 in the binary system. To make binary numbers easier to work with, two different shorthand notations have been defined. In both of these, instead of working with each bit individually, the numbers are collected into subgroups, each of which is assigned a single digit in an alternative numbering system.

### **Octal Numbers**

Consider the binary number 11110100, which is 244 in decimal. Instead of looking at each bit individually, chop them into groups of three, starting from the right: 11110100 becomes (11)(110)(100). Each of those groups has three bits, so each can

have 23 values: from 0 to 7. In this case,  $(1)(1)(110)(100) = (3)(6)(4)$ , or 364 in the *octal* or *base 8* numbering system (see Figure 4-2). As with binary, octal numbers are the same as decimal numbers, except that they use base 8 instead of base 10. So 364 in octal is just  $3 \times 64 + 6 \times 8 + 4$ , or 244. As you can see, octal is a lot less cumbersome than binary, especially when dealing with larger numbers. In the decimal system, 1,000,000 is 3641100 in octal. Compare that with 11110100001001000000 in binary.



**Figure 4-2: Binary, octal, and hexadecimal number representations** A binary number can be represented in octal form by grouping its bits into sets of three, or in hexadecimal by using sets of four bits. These base 8 and base 16 numbers are far shorter than binary numbers, and hence much easier to work with.

## Hexadecimal Numbers

Octal numbers were at one time quite commonly used, but are much less popular today. The problem with octal is that it divides bits into groups of three, but sets of binary numbers typically use a number of bits that is a multiple of *four*.

*Hexadecimal* or the *base 16* numbering system is an alternative method that works like octal, but uses groups of four. Since there are 4 bits in each group, each can have one of 16 values. Hexadecimal is commonly called *hex* for short.

**KEY CONCEPT** Regular numbers are called *decimal numbers* because they are built upon the base 10 system of mathematics. Computers use collections of one or zero bits called *binary numbers*, which can be treated just like regular numbers except that each digit can only be 0 or 1 instead of 0 to 9. Bits in a binary number can be expressed as *octal numbers* by grouping three bits into an *octal digit* that ranges from 0 to 7, or taking sets of four bits to create a single *hexadecimal digit* from 0 to 15. To represent the values 10 through 15 in hexadecimal numbers using a single character, you use the letters A through F.

**NOTE** The term hexadecimal was not the first name used for base 16 numbers in computing. Originally, these were called *sexadecimal numbers*. This is actually the correct term, since Latin prefixes (*sex-*) are normally used for numbers, not Greek ones (*hexa-*). However, in the early 1950s, IBM decided that the word *sexadecimal* was just a little too provocative for their tastes, so they changed it to *hexadecimal*. IBM being IBM—especially back then—meant everyone else followed suit.

Now back to the previous example: 11110100 in binary, 244 in decimal. Next you divide this into groups of four to get (1111)(0100). The binary value 1111 is 15, and 0100 is 4, so you have (15)(4). You need to be able to represent 15, but you only have ten numerals. To solve this problem, the values 10, 11, 12, 13, 14, or 15 in hexadecimal are represented by the letters A, B, C, D, E, and F, respectively. So 11110100 in binary is (15)(4), or F4 in hexadecimal (also shown in Figure 4-2).

Hexadecimal numbers are in some ways even less intuitive than binary ones (it takes some practice to get used to thinking of letters as numbers). Still, hexadecimal is particularly useful as a way to compactly represent binary information. Where 1,000,000 in decimal numbers is 11110100001001000000 in binary, it is only F4240 in hexadecimal numbers—even shorter than the decimal number, since 16 is larger than 10. Also, a single byte has 8 bits, so it can be represented using only two hexadecimal digits. This is why hexadecimal numbers are widely used in computing and networking. For example, you will often see hexadecimal numbers used as network addresses or representing different types of information in frame or packet formats.

**NOTE** *If you see a number that has a letter from A to F in it, you know it is a hex number, but not all hex numbers use these letters. Hex numbers are usually displayed in a special notation, to avoid confusing them with decimal numbers. That notation is either a prefix of 0x or a suffix of h (sometimes both). Thus, the number 54 is just 54, but 0x54 is 54 in hexadecimal numbers, which is  $5 \times 16 + 4$ , or 84 in decimal numbers. Be sure to watch for these representations.*

## Decimal, Binary, Octal, and Hexadecimal Number Conversion

Because people and computers speak different number languages, it is often necessary to convert numbers from one system to another. The easiest way to perform the conversion is with a scientific calculator. However, there will be cases for which you need to perform the conversion by hand.

**NOTE** *If you don't have a scientific calculator, the Windows Calculator program is a reasonable facsimile. Open it, go to the View menu, and change the setting from Standard to Scientific. Click the button next to a numbering system. Then enter a number, and if you click a button next to a different numbering type, the number will be converted for you. There are similar tools for UNIX and Mac OS.*

### Binary, Octal, and Hexadecimal Conversions

To convert between binary, octal, and hex, remember that each octal digit is three binary digits, and each hexadecimal digit is four binary digits. To perform the conversion, group the digits, and convert each group into an octal or hex digit. To convert from hex or octal to binary, convert each hex or octal digit into a set of bits. Table 4-3 shows the conversions from each of the octal and hexadecimal single-digit values to binary (with decimal digits thrown in for convenience).

**Table 4-3:** Binary, Octal, and Hexadecimal Digit Conversion

Binary Digits	Octal Digit	Hexadecimal Digit	Decimal Digit
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6

*(continued)*

**Table 4-3:** Binary, Octal, and Hexadecimal Digit Conversion (continued)

Binary Digits	Octal Digit	Hexadecimal Digit	Decimal Digit
0111	7	7	7
1000	-	8	8
1001	-	9	9
1010	-	A	-
1011	-	B	-
1100	-	C	-
1101	-	D	-
1110	-	E	-
1111	-	F	-

Here are some examples:

**Binary to Octal** Start with the binary number 110101001010. Divide this into groups of three: (110)(101)(001)(010), and then convert each group to a number from 0 to 7 (which is easy to do in your head if you practice a bit). The result is (6)(5)(1)(2), or 6512 octal.

**Hexadecimal to Binary** Start with the hex number 0x4D1B. Convert each digit as given in Table 4-3. Now you have 0x4D1B = (0100)(1101)(0001)(1011), or 0100110100011011.

### **Conversion from Binary, Octal, or Hexadecimal to Decimal**

Conversions to and from decimal are more complicated, because 2, 8, and 16 are powers of two but ten is not. Of the two directions, conversions *to* decimal are easier: You take the value of each binary, octal, or hexadecimal digit, convert it to decimal, and then multiply it by the power of 2, 8, or 16 represented by the digit's place in the number. Then you add all the numbers together. I did this with the example of the decimal number 211 (see Table 4-2).

Table 4-4 shows the hexadecimal number 0x830C converted to decimal (octal uses a similar process). Read the table from left to right, top to bottom; each digit's value is multiplied by the appropriate power of 16 and added together, yielding the decimal result of 33,548.

**Table 4-4:** Hexadecimal to Decimal Number Conversion

Hexadecimal Number	8	3	0	C
Decimal Value of Digit	8	3	0	12
Power of 16	$16^3$	$16^2$	$16^1$	$16^0$
Value of Digit Place	4096	256	16	1
Value for This Number	$8 \times 4096 = 32768$	$3 \times 256 = 768$	$0 \times 16 = 0$	$12 \times 1 = 12$
Running Sum (from left to right)	32768	$32768 + 768 = 33536$	33536	$33536 + 12 = 33548$

## **Conversion from Decimal to Binary, Octal, or Hexadecimal**

Conversions *from* decimal requires you to perform the opposite of the previous calculation: You divide and subtract instead of multiply and add.

### **Conversion from Decimal to Binary**

The easiest of the three conversions from decimal is to binary. Because the maximum value of each digit is 1, there is no dividing, just subtraction. To perform the conversion, do the following:

1. Find the largest power of two that is smaller than the number.
2. Put a 1 in the digit place for that power of two and subtract that power of two from the decimal number.
3. Repeat steps 1 and 2 until you are reduced to zero.

This is easier to explain using an example and a table. Let's convert the decimal number 689, as shown in Table 4-5. Again, read the table starting from the upper left, and going down and then across. You start by noticing that 1024 is not less than or equal to 689, so the 1024s place gets a 0. In the next place, 512 is less than 689, so you make the 512s place a 1 and subtract 512 from 689 to leave 177. The calculation continues, before it eventually shows that the 689 decimal is 1010110001 binary.

**Table 4-5:** Decimal to Binary Number Conversion

Decimal Value Before Considering This Digit Place	689	689	177	177	49	49	17	1	1	1	1
Power of Two	$2^{10}$	$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Value of Digit Place	1024	512	256	128	64	32	16	8	4	2	1
Value of Digit Place Equal to or Less Than Current Decimal Number?	No	Yes	No	Yes	No	Yes	Yes	No	No	No	Yes
Subtraction Step	Skip	$689 - 512 = 177$	Skip	$177 - 128 = 49$	Skip	$49 - 32 = 17$	$17 - 16 = 1$	Skip	Skip	Skip	$1 - 1 = 0$
Binary Digits	0	1	0	1	0	1	1	0	0	0	1

### **Conversion from Decimal to Octal or Hexadecimal**

The process for octal and hexadecimal is almost the same, except that you must divide by powers of two instead of just subtracting, as shown here:

1. Start with the highest power of 16 (hexadecimal) or 8 (octal) that is smaller than the number.
2. Divide the decimal number by that power, keeping only the integer part of the result.

3. Keep the remainder after the division is done.
4. Repeat steps 1 through 3 until you get to the ones place, and then enter whatever is left after the higher digits were done.

Table 4-6 shows the same example as Table 4-5, but goes from decimal to hexadecimal instead of decimal to binary: 689 in decimal is 0x2B1 in hexadecimal.

**Table 4-6:** Decimal to Hexadecimal Number Conversion

Decimal Value Before Considering This Digit Place	689	689	177	1
Power of 16	$16^3$	$16^2$	$16^1$	$16^0$
Value of Digit Place	4096	256	16	1
Value of Digit Place Smaller Than Current Decimal Number?	No	Yes	No	n/a
Division Step	Skip	$689/256 = 2.691$ (use 2 for this digit)	$177/16 = 11.0625$ (use B for this digit)	n/a
Remainder After Division	Skip	177	1	n/a
Hexadecimal Digits	0	2	B	1

## Binary, Octal, and Hexadecimal Arithmetic

We use arithmetic every day to give us the information we need to make decisions. Like us, computers perform arithmetic operations constantly as part of their normal operation, except that computers use binary numbers to perform their calculations incredibly fast.

Binary, octal, and hexadecimal numbers are essentially different representations of numbers, and as such they are not really much different than decimal numbers; they simply have a different number of values per digit. In a similar vein, doing arithmetic with binary, octal, or hexadecimal numbers is not that different from the equivalent operations with decimal numbers. You just have to keep in mind that you are working with powers of 2, 8, or 16, instead of 10, which isn't always easy.

As with number system conversions, calculators are usually the way to go if you need to do math with binary, octal, or hexadecimal numbers. If your calculator does math with only decimal numbers, you can use the trick of converting the numbers to decimal, and then performing the operation and converting the result. However, you can fairly easily do the same addition, subtraction, multiplication, and division on binary, octal, or hexadecimal numbers that you would with decimal numbers by using the Windows Calculator program.

Computers often need to perform multiplication and division operations on binary numbers, but people working with computers don't often perform these operations. Addition and subtraction are much more common operations (especially addition), and they have the added bonus of being much easier to explain. You probably won't need to do this type of arithmetic that often, but it's good to understand it. I'll provide a couple of examples to give you the general idea.

## **Binary Arithmetic**

Let's start with binary. Adding binary numbers is the same as adding decimal ones, except that you end up doing *a lot* of the carrying of ones since there are so few values allowed per digit. Table 4-7 shows an example, with one digit in each column; read it from right to left and top to bottom, just as you would usually do with a manual addition. You start by adding the 1 in the ones place from the first number with the 1 in that place from the second number, thereby yielding a raw digit sum of 2. This means the result for the ones digit is 1, and you carry a 1 to the twos place. You continue with this process until you have added all the digits.

**Table 4-7:** Binary Addition

Carry		1	1			1	1	—
First Binary Number	1	0	1	1	0	0	1	1
Second Binary Number	0	0	1	1	1	0	0	1
Raw Digit Sum	1	1	3	2	1	1	2	2
Result	1	1	1	0	1	1	0	0
Carry to Next Higher Digit			1	1			1	1

## **Octal and Hexadecimal Arithmetic**

Octal and hexadecimal are pretty much the same, except that you carry the number if the sum in a particular digit exceeds either 8 or 16, respectively. Hexadecimal is more common, and more interesting, so let's examine how to add two hexadecimal numbers. While performing the operation, you will need to convert single-digit hexadecimal numbers to decimal and back again, but this isn't too difficult.

The example shown in Table 4-8 should be read from right to left. You start by adding 8 (decimal 8) to A (decimal 10) in the ones place. This yields a raw sum of 18, from which you carry 16 as a 1 to the 16s place and leave a result of 2. You add this 1 to the D (value 13) and E (14 value) of the 16s place. This is a total of 28, leaving 12 (C in hexadecimal), and you carry a 1 to the 256s place. This continues until you are left with a sum of 6DC2h.

**Table 4-8:** Hexadecimal Addition

Carry		1	1		
First Hex Number	2	C	D		8
Second Hex Number	4	0	E		A
Raw Digit Sum	2+4 = 6	1+12+0 = 13	1+13+14 = 28	8+10 = 18	
Result	6	D	C		2
Carry to Next Higher Digit			1		1

## Boolean Logic and Logical Functions

You'll recall that every bit in a computer system can hold a value of either 1 or 0, representing the basic on or off states inherent in a binary digital system, and that you can interpret these on or off values as true or false states, respectively. These values can represent various logical conditions within a system, and you can use various logical operations to manipulate and combine these values to represent more complex logical states.

British mathematician George Boole (1815–1864) was one of the pioneering users of binary values in logical equations, and in recognition of his contribution we call this *boolean logic*.

### Boolean Logical Functions

Boolean logic defines a number of *boolean logical functions*, which are sometimes called *operators*. Each of these functions uses a logical algorithm to compute an output value based on the value of one or more inputs. The algorithm determines when the output is true, based on the combination of true and false values the inputs take. Thus, the table that shows the inputs and outputs for a logical function is called a *truth table*. Each of the logical functions is analogous to a real-world logical operation that you can use to define various logical situations (as you will soon see).

#### NOT

Consider the simplest function: *NOT*. As you might expect, this is just a negation; the output is the opposite of the input. The NOT function takes only one input, so it is called a *unary* function or operator. The truth table for NOT is shown in Table 4-9. As you can see, the output is true when the input is false, and vice versa.

**Table 4-9:** NOT Operator Truth Table

Input	Output
False	True
True	False

The NOT function logically represents the opposite of a condition. For example, suppose you have a bit called B1 whose logical meaning is that when the bit is true, a particular pixel on a screen is lit up. Then the *boolean expression* NOT B1 would be the opposite: It would be false when the pixel is lit up, and thus true only when the pixel is *not* lit up.

Since true and false values are represented in computers by 1 or 0 values, boolean logic is often expressed in terms of ones and zeros, instead of true and false. The circuits inside computer processors and other devices manipulate one and zero bits directly using these functions. In some (but not all) cases, they interpret one and zero as true and false, but in either case, the two representations are functionally equivalent.

Table 4-10 shows the same truth table as Table 4-9, but using bit values. Each true is represented as a 1, and each false is represented as a 0.

**Table 4-10:** NOT Operator Truth Table (Using Bit Values)

Input	Output
0	1
1	0

### AND and OR

The two other primary boolean functions that are widely used are *AND* and *OR*. The output of an AND function is true only if its first, and second, and third inputs and so on are true. The output of an OR function is true if the first input is true *or* the second input is true, and so on.

Both AND and OR can have any number of inputs, with a minimum of two. Table 4-11 shows the truth table for the AND function, with two inputs. You can see that the output is a 1 only when both inputs are 1, but it's 0 otherwise.

**Table 4-11:** AND Operator Truth Table

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Like NOT, AND represents a logical operation similar to how we use the word *and* in our everyday speech. For example, at lunchtime, I might say to a colleague, “Let’s go out for lunch *and* stop at the post office.”

The truth table for the OR function (again with two inputs) is shown in Table 4-12. Here, the output is 1 whenever a 1 appears in at least one input, not necessarily both as in the previous table.

**Table 4-12:** OR Operator Truth Table

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

Interestingly, unlike AND, the OR function does *not* have the same meaning as what we take the word *or* to mean in everyday English. In boolean, the word OR means that the output is true as long as *any* of the inputs is true.

## Exclusive-OR (XOR or EOR)

A modification of OR called *Exclusive-OR* (abbreviated either *XOR* or *EOR*) represents the way we normally use *or* in the real world. Its output is only true if one input or the other is true, but *not both*. The truth table for XOR is as shown in Table 4-13. Notice the difference between this table and Table 4-12: The output is 0 in the case where both inputs are 1.

**Table 4-13:** Exclusive OR (XOR) Operator Truth Table

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

## Combining Boolean Expressions

The functions described earlier can also be combined arbitrarily to produce more complex logical conditions. For example, when searching the Web, you might enter “cheese AND (cheddar OR swiss) NOT wisconsin” into a search engine. In response, the search engine might return pages that contain the word *cheese* and the word *cheddar* or *swiss* (or both), but pages that do *not* contain the word *wisconsin*.

Boolean functions are important because they are the building blocks of much of the circuitry within computer hardware. The functions are implemented as tiny *gates* that are designed to allow electrical energy to flow only to the output based on certain combinations of inputs as described by the truth tables for functions like NOT, AND, OR, and others. In networking, boolean logic is important for describing certain conditions and functions in the operation of networks. Boolean functions are also very important because they are used to set, clear, and mask strings of binary digits, which I will explore in the next section.

**KEY CONCEPT** Boolean logic is a system that uses boolean functions to produce output based on varying conditions in input data. The most common boolean functions are as follows: NOT, which produces output that is the opposite of its input; AND, which is true only if all of its inputs are true; OR, which is true if any of its input is true; and XOR, which is true only if exactly one of its inputs is true (that is, if the inputs are different). These functions can be used in boolean logic expressions that represent conditional states for making decisions, and they can also be used for bit manipulation.

## Bit Masking (Setting, Clearing, and Inverting) Using Boolean Logical Functions

The boolean functions NOT, AND, OR, and XOR describe different ways that logical expressions can be used to manipulate true and false values to represent both simple and complex decisions or conditions. However, these functions can

also be used in a more mundane manner to allow the direct manipulation of binary data. This use of boolean logic is very important in a number of different applications in networking.

You should recall that when you give a bit a value you *set* the bit, and when you give it a value of 0, you *reset* or *clear* it. In some situations bits are handled individually and are set or cleared simply by assigning a 0 or 1 value to each bit. However, it is common to have large groups of bits that are used collectively to represent a great deal of information, whenever many bits need to be set or cleared at once. In this situation, the boolean functions come to the rescue.

### **Setting Groups of Bits with OR**

You can set bits en masse with the OR function. Recall that an OR's output is true (equal to 1) if any of its inputs are true (equal to 1). Thus, if you OR a bit with a value known to be 1, the result will always be 1, no matter what the other value is. In contrast, if you OR with a 0, the original value, 1 or 0, is not changed.

By using a string with 0s and 1s in particular spots, you can set certain bits to 1 while leaving others unchanged. This procedure is comparable to how a painter *masks* areas that he does not want to be painted, using plastic or perhaps masking tape. Thus, the process is called *masking*. The string of digits used in the operation is called the *bit mask*, or simply the *mask*.

For example, suppose you have the 12-bit binary input number 101001011010, and you want to set the middle six bits to be all ones. To do this, you OR the number with the 12-bit mask 00011111000. Table 4-14 shows how this works with the changed bits in the result in bold—you simply OR each bit in the input with its corresponding bit in the mask:

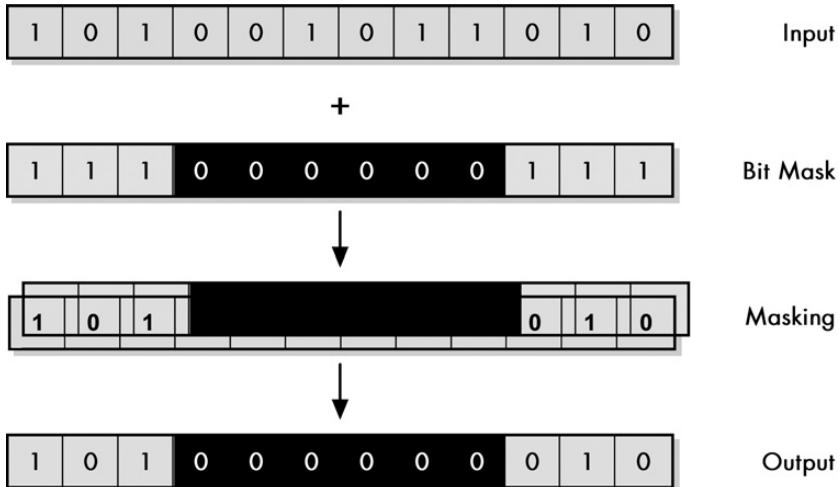
**Table 4-14:** Setting Bits Using an OR Bit Mask

Input	1	0	1	0	0	1	0	1	1	0	1	0
Mask	0	0	0	1	1	1	1	1	1	0	0	0
Result of OR Operation	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>						

### **Clearing Bits with AND**

To clear a certain pattern of bits, you perform a similar masking operation, but using the AND function instead. If you AND a bit with 0, it will clear it to 0, regardless of what the bit was before, while ANDing with 1 will leave the bit unchanged. For example, to clear the middle six bits in Table 4-14, you AND with the reverse bit mask, 111000000111.

Table 4-15 and Figure 4-3 show how a bit mask can be used to clear certain bits in a binary number while preserving others. Each 1 represents a “transparent” area that keeps the corresponding input bit value, while each 0 is a bit where the original value is to be cleared. After performing an AND on each bit pair, the first three and last three bits are preserved, while the middle six, since they were each ANDed with 0, are forced to 0 in the output.



**Figure 4-3: Clearing bits using an AND bit mask** Applying a bit mask to an input binary number using the AND function clears to 0 all bits where the mask bit was 0 and leaves alone bits where the mask was 1.

**Table 4-15:** Clearing Bits Using an AND Bit Mask

Input	1	0	1	0	0	1	0	1	1	0	1	0
Mask	1	1	1	0	0	0	0	0	0	1	1	1
Result of AND Operation	1	0	1	0	0	0	0	0	0	0	1	0

You can also look at this clearing function a different way. You are clearing the bits where the mask is a 0, and in so doing selecting the bits where the mask is a 1. Thus, ANDing with a bit mask essentially means that you keep the bits where the mask is a 1 and remove the bits where it is a 0.

### Inverting Bits with XOR

There are also situations in which you want to *invert* some bits; that is, change a 1 value to a 0, or a 0 value to a 1. To do this, you use the XOR function. While this is not as intuitive as masking, if you refer to the XOR truth table (Table 4-13) you will see that if you XOR with a 1, the input value is flipped, while XORing with a 0 causes the input to be unchanged. To see how this works, use the same input example and invert the middle six bits, as shown in Table 4-16.

**Table 4-16:** Inverting Bits Using an XOR Bit Mask

Input	1	0	1	0	0	1	0	1	1	0	1	0
Mask	0	0	0	1	1	1	1	1	1	0	0	0
Result of XOR Operation	1	0	1	1	1	0	1	0	0	0	1	0

In the world of networking, bit masking is most commonly used to manipulate addresses. In particular, masking is perhaps best known for its use in differentiating between the host and subnet (subnet) portions of Internet Protocol (IP) addresses, a process called *subnet masking* (see Chapter 18, which discusses IP subnet addressing).

**NOTE** Masks are often expressed in either hexadecimal or decimal notation for simplicity, as shown in the IP subnetting summary tables in Chapter 18. However, the masks are always applied in binary, as described previously. You should convert the mask to binary if you want to see exactly how the masking operation will work.

**KEY CONCEPT** The properties of the OR and AND boolean functions make them useful when certain bits of a data item need to be set (changed to 1) or cleared (changed to 0). This process is called *bit masking*. To set bits to 1, a mask is created and used in a bit-by-bit OR function with the input. When the mask has a value of 1, the bit is forced to a 1, while each 0 bit leaves the corresponding original bit unchanged. Similarly, a mask used with the AND function clears certain bits; each 1 bit in the mask leaves the original bit alone, while each 0 forces the output to 0. Finally, XOR can be used to invert selected bits using a mask.

# PART I-2

## **THE OPEN SYSTEMS INTERCONNECTION (OSI) REFERENCE MODEL**

Models are useful because they help us understand difficult concepts and complicated systems. When it comes to networking, there are several models that are used to explain the roles played by various technologies and how they interact. Of these, the most popular and commonly used is the Open Systems Interconnection (OSI) Reference Model. The OSI Reference Model makes it easier for networks to be analyzed, designed, built, and rearranged, by allowing them to be considered as modular pieces that interact in predictable ways, rather than enormous, complex monoliths.

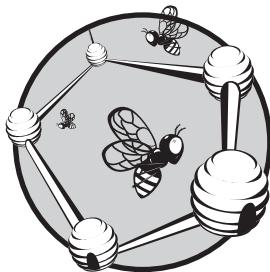
You'll find that it's nearly impossible to read a lot about networking without encountering discussions that presume at least some knowledge of how the OSI Reference Model works. This is why I strongly advise that if you are new to the OSI Reference Model, you read this part carefully. While it is all arguably background material, this information will give you a foundation for understanding networks, as well as make the rest of the book easier to follow. If you are quite familiar with the OSI Reference Model, you may wish to skip this part or just skim through it.

This part is geared to a discussion of networks and internetworks in general, and not specifically to the TCP/IP protocol suite. Therefore, not all of the material in this section is directly relevant to learning about TCP/IP, although much of it is. You may also wish to refer to Part I-3, which includes a discussion of how the TCP/IP and OSI models compare.

In this part, I describe the OSI Reference Model in detail. I begin with a discussion of some general concepts related to the OSI Reference Model and networking models overall. I then describe each of the seven layers of the OSI Reference Model. I conclude with a summary chapter that includes a useful analogy to help you understand how the reference model works to explain the interaction of networks on multiple levels. That chapter also presents a reference table of the layers and their respective functions.

# 5

## GENERAL OSI REFERENCE MODEL ISSUES AND CONCEPTS



The idea behind the OSI Reference Model is to provide a framework for both designing networking systems and explaining how they work. As you read about networking, you will frequently find references to the various levels, or *layers*, of the OSI Reference Model. However, before I can properly discuss the actual OSI model layers, you need to understand the model as a whole.

In this chapter, I introduce the OSI Reference Model and provide some useful background information to help you understand it. I begin with a brief history of the model, including a look at its development and goals. I then introduce networking models in general terms, describing why they are beneficial and how they can best be used. The bulk of the chapter contains important OSI model concepts, which will help you begin to really understand the way model works, the terminology used to describe it, and how it can be of value in explaining the operation of networking technologies.

## History of the OSI Reference Model

A look at the origins of the OSI Reference Model takes us back to several issues related to standards and standards organizations that were discussed in Chapter 3. The idea behind the creation of networking standards is to define widely accepted ways of setting up networks and connecting them together. The OSI Reference Model represented an early attempt to get all of the various hardware and software manufacturers to agree on a framework for developing various networking technologies.

In the late 1970s, two projects began independently with the same goal: to define a unifying standard for the architecture of networking systems. One was administered by the *International Organization for Standardization (ISO)*, while the other was undertaken by the *International Telegraph and Telephone Consultative Committee*, or *CCITT* (the abbreviation is from the French version of the name). These two international standards bodies each developed a document that defined similar networking models.

In 1983, these two documents were merged to form a standard called *The Basic Reference Model for Open Systems Interconnection*. That's a mouthful, so the standard is usually referred to as the *Open Systems Interconnection Reference Model*, the *OSI Reference Model*, or even just the *OSI model*. It was published in 1984 by both the ISO, as standard ISO 7498, and the renamed CCITT (now called the *Telecommunications Standardization Sector of the International Telecommunication Union or ITU-T*) as standard X.200. (Incidentally, isn't the new name for the CCITT much catchier than the old one? Just rolls off the old tongue, doesn't it?)

One interesting aspect of the history of the OSI Reference Model is that the original objective was *not* to create a model primarily for educational purposes, even though many people today think that this was the case. It was intended to serve as the foundation for the establishment of a widely adopted suite of protocols that would be used by international internetworks—basically, what the Internet became. This was called, unsurprisingly, the *OSI protocol suite*.

However, things didn't quite work out as planned. The rise in popularity of the Internet and its TCP/IP protocols met the OSI protocol suite head on, and in a nutshell, TCP/IP won. Some of the OSI protocols were implemented, but as a whole, the OSI protocols lost out to TCP/IP when the Internet started to grow.

The OSI model itself, however, found a home as a device for explaining the operation of not just the OSI protocols, but networking in general. It's used widely as an educational tool—much as I use it myself—and it's also used to help describe interactions between the components of other protocol suites and even hardware devices. Although most technologies were not designed specifically to meet the dictates of the OSI model, many are described in terms of how they fit into its layers. This includes networking protocols, software applications, and even different types of hardware devices, such as switches and routers. The model is also useful to those who develop software and hardware products because it clarifies the roles performed by each of the components in a networking system.

**KEY CONCEPT** The *Open Systems Interconnection Reference Model* (OSI Reference Model or *OSI model*) was originally created as the basis for designing a universal set of protocols called the *OSI protocol suite*. This suite never achieved widespread success, but the model became a very useful tool for both education and development. The model defines a set of layers and a number of concepts for their use that make understanding networks easier.

## General Reference Model Issues

Let's discuss some of the basic issues related to reference models. In part, I want to explain why I place so much emphasis on the OSI model, even going so far as to build much of this book's organization around this model and its layers. I also want you to understand why the model is important, and how it benefits networking not only on a conceptual level, but in reality.

In the topics that follow, I describe several issues that relate to reference models in general terms, and of course, to the OSI Reference Model specifically. I begin with an overview of why networking models are beneficial and why it is important for you to understand how the OSI model works. I then talk about how best to use the model and contrast it with some "real-world" network architectures and protocol stacks.

### ***The Benefits of Networking Models***

Networking is complicated, and special pains must be taken to try to *simplify* it. One of the ways in which networking technology is made easier to understand is by splitting it into pieces, each of which plays a particular role or is responsible for a specific job or function.

However, if this is to be done, you must have a way of ensuring that these various pieces can interoperate; that is, each must know what is expected of it and also what it can expect from the other pieces. This is one of the important roles of networking models. They split the multitude of tasks required to implement modern networks into smaller chunks that can be more easily managed. Just as importantly, they establish "walls" between those pieces and rules for passing information over those walls.

A good analogy of a networking model is that of an assembly line at a manufacturer. No company attempts to have one person build an entire car; even if the company did, it wouldn't expect that individual to be able to learn how to do it all at once. The division of labor offers several advantages to a company that builds a complex product, such as an automobile. Generally speaking, these include the following:

**Training and Documentation** It is easier to explain how to build a complex system by breaking the process into smaller parts. Training can be done for a specific job without everyone needing to know how everything else works.

**Specialization** If everyone is responsible for doing every job, no one gets enough experience to become an expert at anything. Through specialization, certain individuals develop expertise at particular jobs.

**Easier Design Modification and Enhancement** By separating the automobile into systems as well as the particular jobs required to build those systems, you can make changes in the future more easily. Without such divisions, it would be much more difficult to determine what the impact might be of a change, which would serve as a disincentive for innovation.

**Modularity** This is related to each of the previous items. If the automobile's systems and manufacturing steps are broken down according to a sensible architecture or model, it becomes easier to interchange parts and procedures between vehicles. This saves time and money.

Networking models yield very similar benefits to the networking world. They represent a framework for dividing up the tasks needed to implement a network by splitting the work into different levels, or *layers*. Hardware and software running on each layer are responsible for interacting with corresponding hardware and software that are running on other devices on the same layer. The responsibilities of each hardware or software element are defined in part by specifically dividing lines between the layers.

As a result, you get all of the benefits listed in the previous points: easier training, specialized capabilities at each layer, improved capabilities for modification, and modularity. Modularity is particularly important, because it allows you to interchange technologies that run at different layers. While no one would try to build a vehicle that is partly a compact sedan, partly an SUV, and partly a motorcycle, there are situations in networking for which you may want to do something surprisingly similar to this. Networking models help make this possible.

**KEY CONCEPT** Networking models such as the *OSI Reference Model* provide a framework for breaking down complex internetworks into components that can more easily be understood and utilized. The model defines networking functions not as a large, complicated whole, but as a set of layered, modular components, each of which is responsible for a particular function. The result is better comprehension of network operations, improved performance and functionality, easier design and development, and the ability to combine different components in a way that's best suited to the needs of the network.

## **Why Understanding the OSI Reference Model Is Important to You**

A lot of networking books and other resources gloss over the OSI Reference Model or relegate it to the back pages of a hard-to-find appendix. The reason usually stated for this is that the OSI model is “too theoretical” and “doesn’t apply to modern networking protocols like TCP/IP.”

This is a misguided notion. While it is certainly true that the OSI model is primarily theoretical, and that networking protocols aren’t always designed to fit strictly within the confines of their layers, it’s *not* true that the OSI model has little applicability to the real world. In fact, it is difficult to read about networking technology today without seeing references to the OSI model and its layers, because the model’s structure helps to frame discussions of protocols and contrast various technologies.

For example, the OSI Reference Model provides the basis for understanding how technologies like Ethernet and HomePNA are similar; it explains how a PC

can communicate using any of several different sets of protocols, even simultaneously; it is an important part of understanding the differences between interconnection devices such as repeaters, hubs, bridges, switches, and routers; and it also explains how many WAN technologies interoperate.

Far from being obsolete, the OSI model layers are now showing up more than ever in discussions of technology. In fact, some protocols are even *named* specifically in terms of their place in the OSI Reference Model! For an example, consider the Layer Two Tunneling Protocol. Also, switches are now commonly categorized as layer 2, layer 3, or even higher-layer switches.

In theoretical discussions, the OSI Reference Model helps you to understand how networks and network protocols function in the real world. It also helps you to figure out which protocols and devices can interact with each other. So I encourage you to read on. It's time well spent.

**KEY CONCEPT** While many people scoff at the notion of studying the OSI Reference Model, understanding it is very helpful in making sense of networking protocols and technologies. The model is theoretical, but its concepts are employed regularly to describe the operation of real-world networks.

## **How to Use the OSI Reference Model**

Although some people tend to downplay the OSI model too much, others go to the opposite extreme. They use it too much, overanalyzing and trying to use it in a way that was never intended.

The most common mistake is made when attempting to try to “make everything fit” into the layered structure of the OSI model. I must confess to falling into this trap myself on occasion. When I first started laying out the structure of this book, I wanted to organize *everything* based on where it fell in terms of OSI model layers. I quickly discovered that this was like attempting to put pegs of various shapes into a board containing only round holes. I had to change my approach. I ended up organizing it based on the OSI layers where it made sense and using a different structure where it did not.

Learn from my experience. A simple rule of thumb is this: Refer to the OSI Reference Model if it helps you make sense of technologies and understand how they work; *don't* use it if it makes things more complicated. In particular, bear the following in mind:

- It can be very hard to figure out where some technologies fall within the model. Many protocols were designed without the OSI model in mind, and they may not fall neatly into one layer or another. Some overlap two or more layers; other protocol suites may have two protocols that share a layer.
- The boundaries between the upper layers (session, presentation, and application) get particularly fuzzy. Some protocols are clearly designed to fit on one of these layers, while others may overlap all three. This is one reason why I do not categorize higher-level protocols by layer. (The OSI Reference Model was designed to account for the fact that differentiating between these layers might not make sense.)

- The OSI Reference Model was designed primarily with LANs in mind. WAN technologies often fit very poorly into the model, with a lot of overlapping and partial layer coverage. However, it's still useful in most cases to look at these protocols in terms of their approximate fit in the OSI model, since parts of WAN technologies are sometimes interchanged.
- The people who design products don't generally worry about ensuring that their latest inventions implement only specific layers of the model. Thus, sometimes new products come out that break the rules and implement functions across more than one layer, which used to be done by multiple devices at the individual layers. This is usually progress—a good thing!

Finally, an observation: I have noticed that people learning about networking—especially those trying to memorize easy answers to difficult questions so they can pass exams—often ask, “At what layer does this piece of hardware operate?” The problem here is not the answer but rather the question, which is simplistic. With the exception of simple physical devices such as connectors and cables, pretty much *all* networking devices operate at many layers. While a router, for example, is usually associated with layer 3, it has two or more device interfaces that implement layers 2 and 1. A better question is what is the *highest* layer at which a device functions?

The bottom line is that the OSI Reference Model is a tool. If you use it wisely, it can be immensely helpful to you. Just remember not to be too inflexible in how you apply it, and you'll be fine.

**KEY CONCEPT** It is just as much a mistake to assign too much importance to the OSI Reference Model as too little. While the model defines a framework for understanding networks, not all networking components, protocols, and technologies will necessarily fall into the model's strict layering architecture. There are cases in which trying to use the model to describe certain concepts can lead to less clarity rather than more. You should remember that the OSI model is a *tool* and should be used accordingly.

## Other Network Architectures and Protocol Stacks

The OSI Reference Model is not the only model used to describe the structure of networks; several other models and systems are used to describe various sets of networking technologies that work together. These don't generally describe theoretical models, but rather groupings of protocols that are actively used in actual networks. They are, therefore, more often called *networking architectures* and *protocol suites* than models.

As you just saw, many technologies and protocols don't “fit” well into the specific layers used in the OSI model. Similarly, most of the protocol suites used in the real world don't fit the OSI model exactly. This happens, of course, because they were developed independently of the OSI model. Still, most of these architectures and suites still use layers—they are just different from the ones that the OSI model uses.

Since the OSI model is referenced so often, it can be very helpful in making sense of other architectures and even comparing them. Regardless of what the individual layers and technologies are called, networking protocol suites all try to

accomplish the same goals in implementing a network. Thus, even though the layers are not the same, they are often comparable.

In the case of TCP/IP, a special model called the DoD (Department of Defense) model or TCP/IP model is usually used in discussions of the suite (see Chapter 8). This model has many similarities to the OSI model, but also some important differences. In other areas in the field of networking, still other models are used, such as the IEEE 802 networking architecture model. These, too, are similar in some ways to the OSI model, but they have their own unique characteristics.

Even within the scope of some individual specific technologies, you can see a layered structure of related protocols. There are technologies that are generally considered to implement a single level of the OSI model, even though they actually have portions that overlap several OSI layers; examples include Ethernet and Asynchronous Transfer Mode (ATM). In fact, some protocols even have *subprotocols* that are layered within the confines of what is considered a single layer under OSI. A good example is the TCP/IP Point-to-Point Protocol (PPP), which, despite the name, is not a single protocol but a protocol suite unto itself (see Part II-1).

## Key OSI Reference Model Concepts

The OSI Reference Model is valuable as a tool for explaining how networks function, and for describing the relationships between different networking technologies and protocols. To accomplish this, the model relies on a number of important concepts and terms, which I'll discuss in the following sections.

I'll begin with a discussion of how the model uses layers. This is perhaps the single most important of all model concepts. I then talk about some of the notation and jargon you are likely to see in general discussions of the model. I define in more detail what *interfaces* and *protocols* are in the context of the model. I then explain the important concept of data encapsulation and the terminology used to refer to messages in the OSI Reference Model: protocol data units (PDUs) and service data units (SDUs). Finally, I connect most of the preceding issues by describing how the various layers work to handle the routing of messages on a theoretical basis.

### OSI Reference Model Networking Layers, Sublayers, and Layer Groupings

The most important OSI Reference Model concept is that of networking *layers*. It's not an exaggeration to say that layers are really the heart of the OSI model—the entire point of the model is to separate networking into distinct functions that operate at different levels. Each layer is responsible for performing a specific task or set of tasks and dealing with the layers above and below it.

The OSI Reference Model is composed of seven conceptual layers, each of which is assigned a number from 1 to 7. The layer number represents the position of the layer in the model as a whole, and indicates how close the layer is to the actual hardware used to implement a network. The first and lowest layer is the *physical layer*, which is where low-level signaling and hardware are implemented. The seventh and highest layer is the *application layer*, which deals with high-level applications employed by users: both end users and the operating system software.

You can see that as you proceed from the first layer to the seventh, you move up the *layer stack* and, in so doing, increase your level of *abstraction*. This means that the higher a layer is in the stack, the more it deals with logical concepts and software, and the less it deals with the hardware of a network and the nuts and bolts of making it work. The first layer is the most concrete, because it deals with the actual hardware of networks and the specific methods of sending bits from one device to another. It is the domain of hardware engineers and signaling experts. The second layer is a bit more abstract but still deals with signaling and hardware. As you proceed through the third, fourth, and subsequent layers, the technologies at those layers become increasingly abstract. By the time you reach the seventh layer, you are no longer dealing with hardware or even operating system concepts very much; you are in the realm of the user and high-level programs that rely on lower levels to do the “heavy lifting” for them.

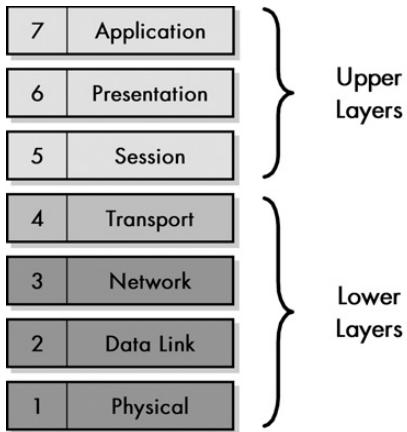
The OSI Reference Model does not formally assign any relationship between groups of adjacent layers. However, to help explain how the layers work, it is common to categorize them into two *layer groupings*:

**Lower Layers (Layers 1, 2, 3, and 4)** As shown in Figure 5-1, the lower layers of the model—*physical*, *data link*, *network*, and *transport*—are primarily concerned with the formatting, encoding, and transmission of data over the network. They don’t care that much about what the data is or what it is being used for; instead, they just want to know about moving it around. They are implemented in both hardware and software, with the transition from hardware to software occurring as you proceed up from layer 1 to layer 4.

**Upper Layers (Layers 5, 6, and 7)** The higher layers of the model—*session*, *presentation*, and *application*—are concerned primarily with interacting with the user and implementing the applications that run over the network. The protocols that run at higher layers are less concerned with the low-level details of how data gets sent from one place to another; they rely on the lower layers to deliver the data. These layers are almost always implemented as software running on a computer or other hardware device.

There are some people who would not necessarily agree with how I have chosen to divide the layers in Figure 5-1. In particular, valid arguments can be made for including the transport layer in the upper layer group, since it is usually implemented as software and is fairly abstract. I place it in the lower layer group because its primary job is still providing services to higher layers for moving data. Really, layer 4 is somewhat of a transition zone and is hard to categorize. Figure 5-1 indicates the special position of layer 4 in the stack.

**KEY CONCEPT** The most fundamental concept in the OSI Reference Model is the division of networking functions into a set of *layers*, from layer 1 at the bottom to layer 7 at the top. As you go up the layer stack, you move away from concrete, hardware-specific functions to ones that are increasingly abstract, until you reach the realm of user applications at layer 7. The seven layers are sometimes divided into groupings: the lower layers (1 through 3) and the upper layers (4 through 7). There is some disagreement on whether layer 4 is a lower or upper layer.



**Figure 5-1: OSI Reference Model layers** The OSI Reference Model divides networking functions into a stack of seven layers, numbered 1 through 7 from the bottom up, and sometimes divided into two layer groupings—the lower layers and the upper layers.

There are also certain OSI layers that have natural relationships to each other. The physical and data link layers, in particular, are closely related. For example, most people talk about Ethernet as a layer 2 technology, but Ethernet specifications really deal with both layer 2 and layer 1. Similarly, layers 3 and 4 are often related; protocol suites are often designed so that layer 3 and 4 protocols work together. Good examples are TCP and IP in the TCP/IP protocol suite, and IPX and SPX in the Novell suite.

In some areas, the layers are so closely related that the lines between them become *blurry*. This is particularly the case when looking at the higher layers; many technologies implement two or even all three of these layers, which is another reason why I feel they best belong in a group together. One important reason why the distinctions between layers 5 through 7 are blurry is that the TCP/IP protocols are based on the TCP/IP model (covered in Chapter 8), which combines the functions of layers 5 through 7 in a single, thick layer.

**KEY CONCEPT** The four lower layers of the OSI model are most often discussed individually, because the boundaries between them are reasonably clear-cut. In contrast, the lines between the session, presentation, and application layers are somewhat blurry. As a result, sometimes protocols span two or even all three of these layers; this is especially true of TCP/IP application protocols, since the TCP/IP model treats layers 5 through 7 as a single layer.

Finally, note that some OSI Reference Model layers are further divided into *sublayers* to help define more precisely the internal details of protocols and technologies at those layers. This is commonly done at the lower layers, especially at the physical layer and the data link layer.

## ***"N"* Notation and Other OSI Model Layer Terminology**

As a theoretical model, the OSI Reference Model comes complete with a set of terminology that is used to describe it and its constituent parts. This is sort of both good news and bad news. The good news is that if you understand this terminology, it can help you comprehend how technologies relate to the model as well as most OSI model discussions in general. The bad news is that the terminology can also increase confusion—especially since it isn't always used consistently.

Here are a few terminology concepts you will often see used to refer to the OSI Reference Model:

**Layer Names and Numbers** The various layers of the OSI Reference Model are referred to in a variety of ways. They may have their names spelled out in full, or they may be abbreviated. They are also often simply referenced by their layer number. So, for example, all of these refer to the same thing: data link layer, Data Link Layer, DLL, L2, layer two, and layer 2. Similarly, you will often see layer names being used as adjectives to describe protocols and technologies. For example, a layer 3 technology is one that operates primarily at the network layer.

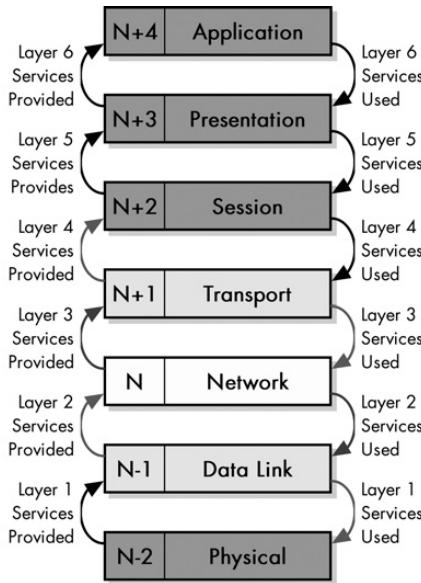
**N Notation** The letter *N* is often used to generically refer to a number within the computer world. With respect to the OSI model, it's common to see this letter used in discussions that relate generically to individual layers without mentioning a specific layer. You will hear terms like *N-functions* and *N-services*, which just refer to the functions and services provided within a particular layer. As another example, you might hear someone say that a particular technology "provides a useful service to the *N+1* layer." This just means it provides a function to the layer above the one at which it operates. Conceptually, every layer but the first and seventh have an *N-1* layer, an *N+1* layer, and so on. If you are looking at the network layer (layer 3), then the *N+2* layer is the session layer (layer 5).

**Protocols and Interfaces** These words have special meaning within the context of the OSI model. A *protocol* represents communication between logical or physical devices at the same layer of the model. An *interface* represents information moving between adjacent layers within the same device. Thus, in *N* notation, protocols represent communication between layer *N* on one device and layer *N* on another device, while interfaces deal with communication between layer *N* and *N+1* or layer *N* and *N-1* on the same device.

**Network Stacks** What do you get when you take a bunch of layers and pile them up on top of each other? You get a *stack*. This term is used to refer to the entire set of layers in a model or suite of technologies, or a partial set. Since each layer has protocols associated with it, this is also sometimes called the *protocol stack*.

**Entities, Functions, Facilities, and Services** These often interchanged, somewhat vague terms refer to specific tasks or jobs performed at various layers in the model. An *N-entity* is a term that refers to a specific operation or job done at layer *N*. A *function* is basically the same thing. *Facilities* and *services* are what a layer provides to the layers above it. This is often expressed in *N*-notation as well: the *N+1* layer often uses a set of *N* services or *N* facilities provided by the *N* layer.

Figure 5-2 serves as a summary of the previous information by showing the relationships between OSI model layers and the terminology used to refer to adjacent layers in the context of any particular layer. Each layer (except layer 7) provides services to the layer above it; each layer (other than layer 1) uses services provided by the layer below it. Another way of saying this is that each layer *N* provides services to layer *N+1* and uses the services of layer *N-1*. Taking the example of layer 3, the network layer, you see that it provides services to layer 4 and uses services of layer 2. From the standpoint of the network layer, the transport layer is layer *N+1* and the data link layer is *N-1*.



**Figure 5-2: OSI Reference Model layer relationships and terminology** Each layer has a relationship with the layer above and below it; here, if the network layer is layer N, it provides services to the transport layer (layer N+1) and uses services of the data link layer (layer N-1).

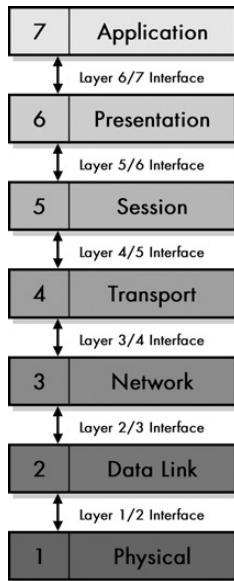
You may have just read all of that and said to yourself, “Why do they bother making this so *complicated* anyway?” Good question. Remember, I *did* say there was bad news here! Now that you know what all of this stuff is about, if you run into it, you won’t be *too* confused.

Fortunately, the use of the previous buzzwords is somewhat limited. Most references are to specific layer names or numbers, and in particular, the N-1 and N+1 stuff is rarely used in discussions of real-world technologies. However, it can be very useful in explaining the model itself, as you will see in some of these terms when you read the rest of this chapter.

### Interfaces: Vertical (Adjacent Layer) Communication

The seven layers of the OSI Reference Model are used to divide the various functions that are required to implement a networking system. On any given device in a network, different software and hardware routines and devices may be functioning on any or all of these layers simultaneously. Because, in general, all of these are supposed to be working together to implement networking functions, there is a need for layers to communicate *vertically* between the layers within a particular host.

In OSI Reference Model parlance, the mechanism for communication between adjacent layers in the model is called an *interface*. Of course, the term *interface* is also used widely in other contexts in the computer and networking worlds, since its generic meaning refers to connecting just about *anything* together. However, when someone talks about an interface between OSI model layers, that person typically refers to the process by which data is passed between layer N of the model and layer N-1 or layer N+1. These relationships are demonstrated in Figure 5-3. For example, the *layer 2/3 interface* is used by a layer 2 and layer 3 protocol to pass data and control information; the *layer 3/4 interface* connects layers 3 and 4 together.



**Figure 5-3: OSI Reference Model interfaces for vertical communication**

In OSI model terminology, an interface is a conduit for communication between adjacent layers in the layer stack.

**NOTE** Remember that not all layers may be implemented in every system or protocol stack in the real world. So it's possible that a process that is technically running at layer 7 might communicate with one running at layer 5. However, I am talking about the theoretical model here.

Vertical communication is done up and down the protocol stack every time anything is sent across the network, and of course, whenever anything is received. This occurs because the higher levels are implemented as logical functions in software; there is no actual physical connection. The higher layers package data and send it down to the lower layers for it to be sent across the network. At the very lowest level, the data is sent over the network. On the receiving end, the process is reversed, with the data traveling back up to the higher layers on the receiving device. I'll discuss this logical interaction between corresponding layers momentarily.

One of the primary goals of the OSI Reference Model is to allow the interconnection of different implementations of various layers. Thus, the intention is to have somewhat autonomous individual layers that you can mix and match—to a point. The only way to make this work is to have well-defined ways that the layers connect together, and that brings me back to the matter of interfaces. Each layer must present a consistent, well-documented interface to the layers above it so that any upper layer implementation can use the lower layer properly.

I'll provide an example from the world of TCP/IP to illustrate what I mean. The heart of the TCP/IP protocol suite is the Internet Protocol (IP). Whenever you use any application on the Internet—email, websites, FTP, chat rooms, and so on—you are indirectly using IP.

However, you never use IP directly—you generally use one of two transport layer (layer 4) protocols: the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) (see Part II-8). A standard interface exists between the network layer and the transport layer in the TCP/IP protocol stack, which defines

how IP is to be used by upper layer protocols; this enables TCP and UDP to interface to it. Similarly, both TCP and UDP present a particular interface to the hundreds of higher-layer protocols and applications that use them at higher layers.

Many different types of communication actually take place between layers. Control information is passed to enable the higher layers to utilize the lower ones, and for the lower ones to pass status and results information back to the higher ones. Data is also passed in both directions across the interface. For transmission, it flows down to the lower layer, which normally results in data encapsulation. Upon reception, the process is reversed, with data being sent back up across the interface from a lower to higher layer.

**KEY CONCEPT** In the OSI Reference Model, an *interface* defines the mechanism for vertical communication between adjacent layers. The existence of well-defined interfaces between layers is what permits a higher layer to use the services of any of a number of lower layers, without requiring knowledge of how those layers are implemented.

### **Protocols: Horizontal (Corresponding Layer) Communication**

Each layer in the OSI Reference Model has a particular role (or roles)—a set of general tasks for which it is responsible. On each system on the network, hardware and software are running at many of the different levels in the model. The routines doing a particular job on Machine A are designed to communicate with similar or complementary ones that are running on Machine B. This *horizontal communication* is the very heart of what networking is about. It is what enables web browsers and web servers to talk, email applications to exchange messages, and so much more.

Of course, all communication types function only if everyone agrees to the same methods of accomplishing it. Each set of rules describing one type of communication is called a *protocol*. You can think of a protocol as a language or a set of instructions. Each function or service of a network has its own language; like human languages, some are similar to each other while others are quite unique.

If you've done any reading at all about networks, you have probably seen the term *protocol* many, many times. Like the word *interface*, the word *protocol* can have many meanings. In fact, it is so fundamental to networking, and used in so many different ways, that I have a discussion devoted to it in Chapter 1.

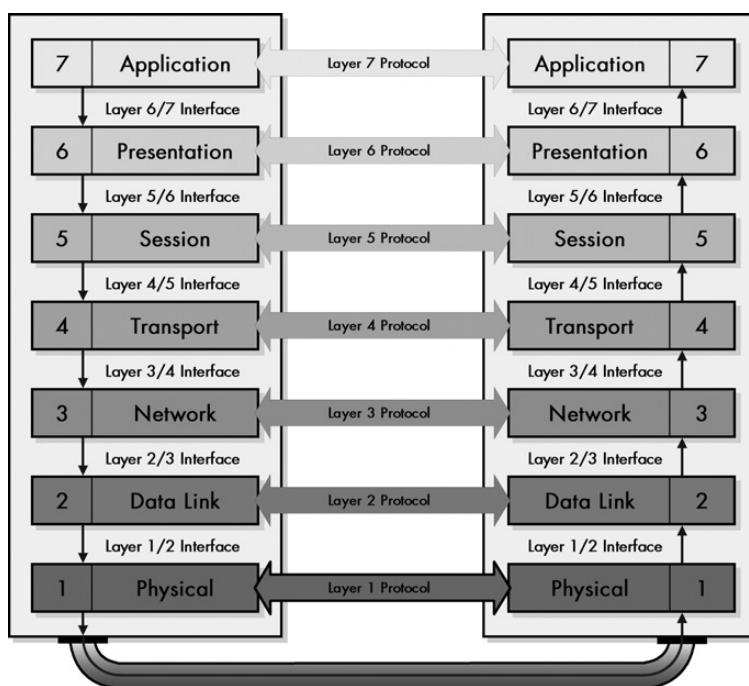
All that aside, you must remember that the OSI Reference Model is intended to be a formal way of describing networks. As such, the term *protocol* has a formal meaning in the context of the model. It refers specifically to a set of communication rules, instructions, and procedures that describe communication between specific software or hardware elements running *at the same layer* on different machines within a network.

Let's consider how these corresponding layers communicate using protocols. First, you'll recall that every layer in the model, except the bottom (physical) layer, is really a program or algorithm running on a computer. There is no way for, say, a web browser and a web server to actually connect together directly—they are just software programs, after all. Instead, the software running at various layers communicates *logically*. That is to say, through the use of software and procedures, a process running at layer 5 on one machine can accomplish *logical communication* with a similar process running at layer 5 on another machine.

Since machines are only physically connected at layer 1, the data on the sending machine must pass down the data through the layers between layer 5 and layer 1 in order for a protocol at layer 5 to function. The data is then transmitted over the physical connection to layer 1 of the other machine and passed up on the protocol stack of the receiving machine to layer 5. This is how the two machines are logically linked at layer 5, even though they have no physical connection at that layer.

Thus, with the exception of the actual physical connection at layer 1, all horizontal communication also requires vertical communication—down the stack on one machine, and then back up the stack on the other. (The communication doesn't always go all the way back up the stack for each connection, however, as in the case of routing, as discussed in the “Indirect Device Connection and Message Routing” section at the end of this chapter.)

Figure 5-4 illustrates how horizontal communication works. As an example, IP is said to be a layer 3 protocol because each device uses IP software to communicate at layer 3. The actual transmission and reception of data occurs only at the lowest, physical layer; higher-layer protocols communicate *logically* by passing data down interfaces until it reaches layer 1, transmitting at layer 1, and then passing the data back up to the appropriate layer at the recipient.



**Figure 5-4: OSI Reference Model protocols: horizontal communication** The term protocol has many meanings; in the context of the OSI Reference Model, it refers specifically to software or hardware elements that accomplish communication between corresponding layers on two or more devices.

**KEY CONCEPT** In the OSI Reference Model, a *protocol* refers specifically to a set of rules or procedures that define communication between software or hardware elements running at the same layer on network devices. Physical layer protocols are responsible for the actual transmission and reception of data at layer 1. Protocols at higher layers pass data down through the layers below them to layer 1 for transmission, then across the network and back up to the corresponding entity at the same layer on the receiving device. The result is that software processes running at say, layer 4 on each of two devices can communicate *logically* as if they were directly connected at layer 4, even though they are not.

## **Data Encapsulation, Protocol Data Units (PDUs), and Service Data Units (SDUs)**

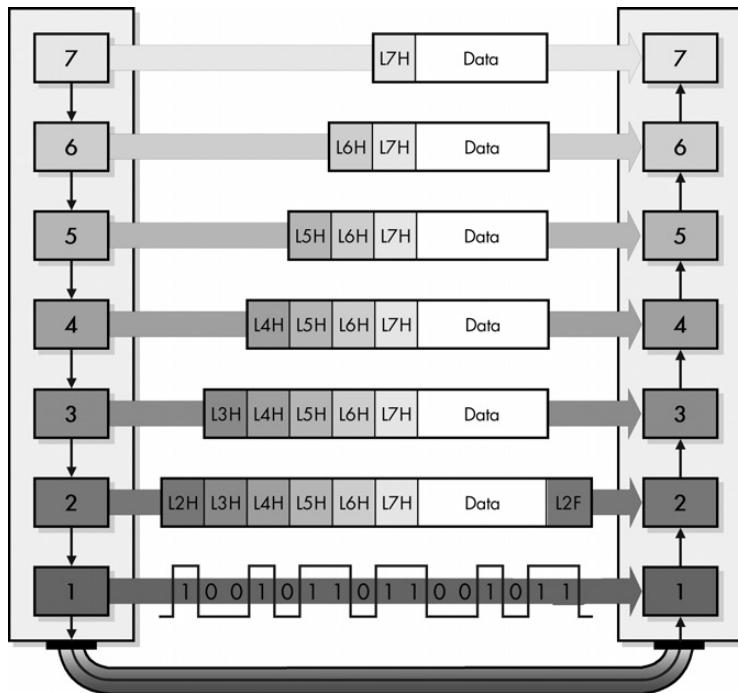
Protocols are what describe the rules that control horizontal communication, that is, conversations between processes that run at corresponding layers within the OSI Reference Model. At every layer (except layer 1), these communications ultimately take the form of some sort of message that is sent between corresponding software elements on two or more devices. Since these messages are the mechanism for communicating information between protocols, they are most generally called *protocol data units (PDUs)*. Each PDU has a specific format that implements the features and requirements of the protocol.

As discussed in the previous section, the communication between layers higher than layer 1 is *logical*; the only hardware connection is at the physical layer. Thus, in order for a protocol to communicate, it must pass down its PDU to the next lower layer for transmission. You've also already seen that, using OSI terminology, lower layers are said to provide *services* to the layers immediately above them. One of the services each layer provides is this function: to handle and manage data received from the layer above.

At any particular layer N, a PDU is a complete message that implements the protocol at that layer. However, when this layer N PDU is passed down to layer N-1, it becomes the *data* that the layer N-1 protocol is supposed to *service*. Thus, the layer N protocol data unit (PDU) is called the layer N-1 *service data unit (SDU)*. The job of layer N-1 is to transport this SDU, which it does by placing the layer N SDU into its own PDU format, preceding the SDU with its own headers and appending footers as necessary. This process is called *data encapsulation*, because the entire contents of the higher-layer message are encapsulated as the data payload of the message at the lower layer.

What does layer N-1 do with its PDU? It passes it down to the next lower layer, where it is treated as a layer N-2 SDU. Layer N-2 creates a layer N-2 PDU containing the layer N-1 PDU and layer N-2's headers and footers. And so the process continues, all the way down to the physical layer. In the theoretical model, what you end up with is a message at layer 1 that consists of application-layer data that is encapsulated with headers and footers from layers 7 through 2.

Figure 5-5 shows a layer 7 PDU consisting of a layer 7 header (labeled L7H) and application data. When this is passed to layer 6, it becomes a layer 6 SDU. The layer 6 protocol prepends to it a layer 6 header (labeled L6H) to create a layer 6 PDU, which is passed to layer 5. The encapsulation process continues all the way down to layer 2, which creates a layer 2 PDU—in this case, shown with both a header and a footer—that is converted to bits and sent at layer 1.



**Figure 5-5: OSI Reference Model data encapsulation** Each protocol creates a protocol data unit (PDU) for transmission, each of which includes headers required by that protocol and data to be transmitted. This data becomes the service data unit (SDU) of the next layer below it.

**KEY CONCEPT** The message used to communicate information for a particular protocol is called its *protocol data unit (PDU)* in OSI model terminology. That PDU is passed down to the next lower layer for transmission; since that layer is providing the service of handling that PDU, it is called the lower layer's *service data unit (SDU)*. The SDU is encapsulated into that layer's own PDU and, in turn, sent to the next lower layer in the stack, proceeding until the physical layer is reached. The process is reversed on the recipient device. In summary, a layer N PDU is a layer N-1 SDU, which is *encapsulated* into a layer N-1 PDU.

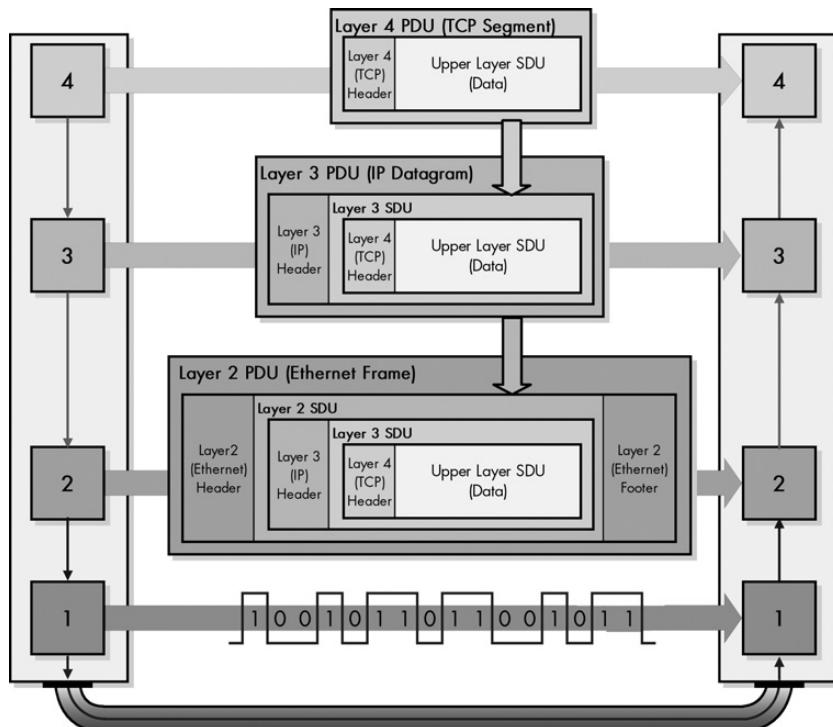
The “N-1, N-2” stuff makes this seem more difficult than it really is, so let’s use a real-world (simplified) example instead. TCP operates at layer 4 of the OSI model. It transmits messages called *segments* that contain data encapsulated from higher-layer protocols. The layer below TCP is IP at layer 3. It receives data from TCP and encapsulates it for transmission.

So, in the formal language of the OSI Reference Model, TCP segments are created as layer 4 PDUs. When passed to IP, they are treated as layer 3 SDUs. The IP software packages these SDUs into messages called *IP packets* or *IP datagrams*, which are layer 3 PDUs. These are passed down to a layer 2 protocol, say Ethernet, which treats IP datagrams as layer 2 SDUs, and packages them into layer 2 PDUs (Ethernet frames), which are sent on to layer 1. (Actually, in some technologies, further encapsulation even occurs at layer 1 prior to transmission.)

On the receiving device, the process of encapsulation is reversed. The Ethernet software inspects the layer 2 PDU (Ethernet frame) and removes from it the layer 2 SDU (IP datagram), which it passes up to IP as a layer 3 PDU. The IP layer removes the layer 3 SDU (TCP segment) and passes it to TCP as a layer 4 PDU. TCP continues the process, going back up the protocol layer stack.

Figure 5-6 shows in more detail how OSI PDUs and SDUs are created and encapsulated. A TCP segment (layer 4 PDU) becomes a layer 3 SDU, which is encapsulated into a layer 3 PDU through the addition of an IP header. This becomes the payload of an Ethernet frame, which is a layer 2 PDU containing an Ethernet header, a layer 2 SDU (the IP datagram), and an Ethernet footer. The receiving device extracts the IP datagram from the Ethernet header and passes it to layer 3; the IP software extracts the TCP segment and passes it up to the TCP software.

This whole matter of encapsulation, passing data up and down the protocol stack, and so on may seem needlessly complex. It also may appear to be rather inefficient; why send a message with so many headers and footers? However, the notion of data encapsulation is critical to creating modular, flexible networks.



**Figure 5-6: OSI Reference Model PDU and SDU encapsulation** Each PDU at one layer of the OSI model becomes an SDU at the next lower layer and is encapsulated into that layer's PDU.

The term *protocol data unit* or PDU is rather formal. You will see it used in standards and sometimes in discussions, but more often than not, you'll encounter the message terms, such as *frame* and *datagram*, as discussed in Chapter 1. Similarly, data encapsulated by these messages is not normally called a *service data unit* or SDU, but rather simply the *message body* or *payload*, as you saw when you looked at

message formatting in Chapter 1. There are cases, however, for which knowing the difference between an SDU and a PDU is important to understanding the technology. One example is the IEEE 802.11 physical layer—the 802.11 standards talk about SDUs and PDUs constantly!

**RELATED INFORMATION** See the OSI Reference Model analogy in the “The Benefits of Networking Models” section earlier in this chapter for an example that compares networking encapsulation to something done in a real-world, nonnetworking context.

## **Indirect Device Connection and Message Routing**

Most of the explanations that I have provided in the other sections of this chapter have discussed the mechanisms by which machines connect to each other over a network *directly*. However, one of the most powerful aspects of networking is that it is possible to create internetworks—networks of networks—that allow devices to be connected *indirectly*. For example, Machine A may send a message to Machine B without really even knowing where it is on the network.

If a message is being sent between devices that are not on the same network, then it must be passed between directly connected networks until it reaches its final destination. The process of transmitting a message from one network to another is called *forwarding*, and the collective process of forwarding from one device to another is *routing*. These concepts are fundamental to all internetworking, including the Internet itself. Every time you access an Internet resource such as a website, you are sending messages that get routed to that site, and the responses you receive get routed back.

**NOTE** Even though the technically correct term for moving a message from one network to an adjacent network is *forwarding*, over time, the term *routing* has come to be used both for a single network-to-network transfer, as well as the overall process of transmitting a message from one device to another.

In the context of the OSI Reference Model, routing is an activity that generally takes place at the network layer, layer 3. You’ll recall that data encapsulation causes a higher-layer message to be surrounded by headers and footers at the lower layers. When a message is routed, here’s what happens:

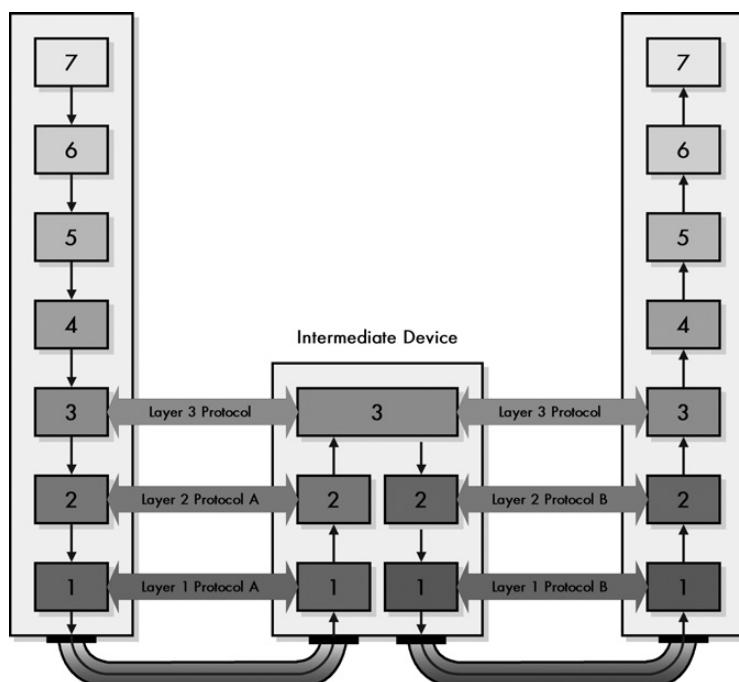
- A high-level application on a machine decides to send a datagram to a distant computer. The datagram is packaged, and then passed down vertically through the protocol stack on the originating machine. Each layer encapsulates the data, as described in the previous section. The datagram is addressed to the final destination device. When the message gets to the lower layers, however, it is not packaged for local delivery directly to its ultimate destination, but rather passed to an *intermediate device*. This is the device that is responsible for routing to that destination network. The message is passed down to the data link and physical layers for transmission to that intermediate device.
- The intermediate device (often called a *router*) receives the message at the physical layer. It is passed up to the data link layer, where it is processed, checked for errors and so on, and the data link layer headers are removed. The resulting packet is passed up to the network layer. There, the intermediate device determines if the destination machine is on its local network, or if it needs to

be forwarded to another intermediate device. It then repackages the message and passes it back *down* to the data link layer to be sent on the next leg of its journey.

- After several potential intermediate devices handle the message, it eventually reaches its destination. Here, it travels back up the protocol stack until it reaches the same layer as the one from the application that generated the message on the originating machine.

The key to this description is that in the intermediate devices, the message travels back up the OSI layers *only to the network layer*. It is then repackaged and sent back along its way. The higher layers are involved only on the source and destination devices. The protocol used at layer 3 must be common across the internetwork, but each individual network can be different. This demonstrates some of the power of layering by enabling even rather dissimilar physical networks to be connected together.

Figure 5-7 shows how routing is accomplished conceptually in the OSI model. The intermediate device connects the networks of the message transmitter and recipient. When data is sent, it is passed up to the network layer on the intermediate device, where it is repackaged and sent back down the stack for the next leg of its transmission. Note that the intermediate device actually has two different layer 1 and 2 implementations—one for the interface to each network. Also note that while the layer 3 protocol must be the same across the internetwork, each network can use different technologies at layers 1 and 2.

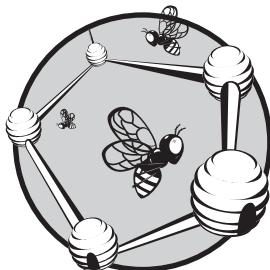


**Figure 5-7: Message routing in the OSI Reference Model** Routing in the OSI model is accomplished using an intermediate device that connects networks at layer 3. Data passes up to layer 3 in that device on one network and then passes back down to layer 1 on another.

**KEY CONCEPT** In the OSI model, the process of *routing* occurs when data is sent not directly from transmitter to ultimate recipient, but indirectly through the use of an intermediate system. That device, normally called a *router*, connects to two or more physical networks, and thus has multiple interfaces to layer 2. When it receives data, the data passes up only to the network layer, where it is repackaged and then sent on the next leg of its journey over the appropriate layer 2 interface.

# 6

## OSI REFERENCE MODEL LAYERS



In this chapter, we look at the individual layers of the OSI Reference Model. Each layer in the OSI model has certain characteristics that define it, and also various protocols normally associated with it. I'll describe how each layer functions in the OSI layer stack, outline the specific types of activities for which each is normally responsible, and provide some examples of the technologies and protocols that reside at each layer. Understanding the nuances of each layer will help you understand all the technologies that use them.

Keep in mind, however, that the descriptions in this section are *generic*. To really comprehend the details of the various layers and how they are used, read the details of the individual protocols that function at each layer later in this book.

**RELATED INFORMATION** Chapter 7 contains summary information that may be helpful to you in understanding the OSI model layers. This includes some common mnemonics for remembering the order of the layers and a summary chart for quickly comparing the layers' key characteristics.

## Physical Layer (Layer 1)

The lowest layer of the OSI Reference Model is layer 1, the *physical layer*; it is commonly abbreviated PHY. This layer is the only one where data is physically moved across the network interface. All other layers perform functions to create messages that implement various protocols, but these messages must all be transmitted down the protocol stack to the physical layer, and they are eventually sent out over the network.

First, a bit of clarification. The name *physical layer* can be a bit problematic because it suggests that this layer relates only to the actual network hardware, which is not the case. While some people say that the physical layer is the network interface cards and cables, this is not actually true. The physical layer defines a number of network functions in addition to interfaces with hardware cables and cards.

People also suggest that all network hardware belongs to the physical layer. Again, this isn't strictly accurate. All hardware must have *some* relation to the physical layer in order to send data over the network, but hardware devices generally implement multiple layers of the OSI model in addition to the physical layer. For example, an Ethernet network interface card performs functions at both the physical layer and the data link layer.

The physical layer technologies deal with the actual ones and zeros that are sent over the network. For example, repeaters, conventional hubs, and transceivers all operate at the physical layer. These devices have no knowledge of the contents of a message; they simply take input bits and send them as output. The physical layer is responsible for the following:

**Hardware Specifications Definition** The details of operation of cables, connectors, wireless radio transceivers, network interface cards, and other hardware devices are generally a function of the physical layer (although also partially the data link layer, layer 2).

**Encoding and Signaling** The physical layer is responsible for various encoding and signaling functions that transform the data from bits that reside within a computer or another device into signals that can be sent over the network.

**Data Transmission and Reception** After encoding the data appropriately, the physical layer actually transmits the data, and of course, receives it. (This applies equally to wired and wireless networks, even if there is no tangible cable in a wireless network.)

**Topology and Physical Network Design** The physical layer is also considered the domain of many hardware-related network design issues, such as local area network (LAN) and wide area network (WAN) topology.

While the physical layer of a network primarily defines the hardware it uses, it is also closely related to the data link layer. Thus, it is not generally possible to define hardware at the physical layer independently from the technology being used at the data link layer. For example, Ethernet is a technology that describes specific types of cables and network hardware, but the physical layer of Ethernet can be isolated

from its data link layer aspects only to a point. Though Ethernet cables are the physical layer, the cables' maximum length is related closely to message format rules that exist at the data link layer.

Furthermore, some technologies perform functions at the physical layer that are normally more closely associated with the data link layer. For example, it is common to have the physical layer perform low-level (bit-level) repackaging of data link layer frames for transmission. Error detection and correction may also be done at layer 1 in some cases, though most people would consider these layer 2 functions.

In many technologies, a number of physical layers can be used with a data link layer. The classic example is Ethernet, for which dozens of different physical layer implementations exist. Each implementation uses the same data link layer (possibly with slight variations).

**KEY CONCEPT** The lowest layer in the OSI Reference Model is the *physical layer*. It is the realm of networking hardware specifications, and is the place where technologies that perform data encoding, signaling, transmission, and reception functions reside. The physical layer is closely related to the data link layer.

Many technologies further subdivide the physical layer into *sublayers* in order to allow different network media to be supported by the same technology, while sharing other functions at the physical layer that are common between the various media. A good example of this is the physical layer architecture used for Fast Ethernet, Gigabit Ethernet, and 10-Gigabit Ethernet.

**NOTE** In some contexts, the physical layer technology that's used to convey bits across a network or communications line is called a *transport method* (not to be confused with the OSI transport layer, layer 4).

## Data Link Layer (Layer 2)

The second-lowest layer (layer 2) in the OSI Reference Model stack is the *data link layer*, often called simply the *link layer*, or abbreviated DLL. The data link layer is where many wired and wireless LAN technologies primarily function. For example, Ethernet, Token Ring, FDDI, and 802.11 (wireless Ethernet or Wi-Fi) are all sometimes called data link layer technologies. The set of devices connected at the data link layer is commonly considered a simple network (as opposed to an internetwork, which is a collection of networks connected at layer 3).

The data link layer is often conceptually divided into two sublayers: *logical link control (LLC)* and *media access control (MAC)*. This split is based on the architecture used in the IEEE 802 Project, which is the IEEE working group responsible for creating the standards that define many networking technologies. By separating LLC and MAC functions, interoperability of different network technologies is made easier, as explained in the discussion of networking models in Chapter 5.

The following are the key tasks performed at the data link layer:

**Logical Link Control (LLC)** Logical link control refers to the functions required for the establishment and control of logical links between local devices on a network. This is usually considered a sublayer; it provides services to the network layer

above it and hides the rest of the details of the data link layer, which allows different technologies to work seamlessly with the higher layers. Most LAN technologies use the IEEE 802.2 LLC protocol to implement this part of the data link layer.

**Media Access Control (MAC)** This refers to the procedures used by devices to control access to the network medium. Since many networks use a shared medium (such as a single network cable, or a series of cables that are electrically connected into a single virtual medium), it is necessary to have rules for managing the medium to avoid conflicts. For example, Ethernet uses the CSMA/CD method of media access control, while Token Ring uses token passing.

**Data Framing** The data link layer is responsible for data framing, which is the final encapsulation of higher-level messages into *frames* that are sent over the network at the physical layer.

**Addressing** The data link layer is the lowest layer in the OSI model that is concerned with addressing. It labels information with a particular destination location. Each device on a network has a unique number that is used by the data link layer protocol to ensure that data intended for a specific machine gets to it properly. This is usually called a *hardware address* (since it is intimately related with low-level hardware) or a *MAC address* (after the MAC function described earlier).

**Error Detection and Handling** The data link layer handles errors that occur at the lower levels of the network stack. For example, a cyclic redundancy check (CRC) field is often calculated based on the frame's contents and then included in it. This can be employed to allow the station receiving data to detect if it was received correctly.

**Physical Layer Standards** The physical layer and the data link layer are very closely related. The requirements for the physical layer of a network are often part of the data link layer standard that describes a particular technology. Certain physical-layer hardware and encoding aspects are specified by the data link layer technology being used. The best example of this is the Ethernet standard, IEEE 802.3, which specifies not just how Ethernet works at the data link layer, but also its various physical layers.

**KEY CONCEPT** The second OSI Reference Model layer is the *data link layer*. This is where most LAN and wireless LAN technologies are defined. Layer 2 is responsible for *logical link control (LLC)*, *media access control (MAC)*, hardware addressing, error detection and handling, and defining physical layer standards. It is often divided into the LLC and MAC sublayers based on the IEEE 802 Project that uses that architecture.

Many types of hardware are associated with the data link layer. Network interface cards typically implement a specific data link layer technology, so they are often called Ethernet cards, Token Ring cards, and so on. There are also a number of network interconnection devices that are said to operate at layer 2 in whole or in part because they make decisions about what to do with data they receive by looking at data link layer frames. These devices include most bridges, switches, and routers, though the latter two also encompass functions performed by layer 3.

Some of the most popular technologies and protocols generally associated with layer 2 are Ethernet, Token Ring, FDDI (plus CDDI), HomePNA, IEEE 802.11, Asynchronous Transfer Mode (ATM), TCP/IP's Serial Line Interface Protocol (SLIP), and TCP/IP's Point-to-Point Protocol (PPP).

## Network Layer (Layer 3)

The third-lowest layer of the OSI Reference Model is the *network layer*. If the data link layer defines the boundaries of what is considered a network, the network layer defines how *internetworks* (interconnected networks) function. The network layer is the lowest one in the OSI model that is concerned with actually getting data from one computer to another even if it is on a remote network; in contrast, the data link layer only deals with devices that are local to each other.

While layers 2 through 6 all act as fences between the layers above and below them, the network layer is particularly important in terms of separating higher and lower-layer functions. It is here that the transition really begins from the more abstract functions of the higher layers—which don't concern themselves as much with data delivery—into the specific tasks required to get data to its destination. (The transport layer continues this abstraction transition as you go up the OSI protocol stack.)

Some of the specific jobs normally performed by the network layer include the following:

**Logical Addressing** Every device that communicates over a network has a logical address associated with it, which identifies the device regardless of its particular location. This is sometimes called a *layer 3* address. For example, on the Internet, the Internet Protocol (IP) is the network layer protocol and every machine has an IP address. Logical addresses are independent of particular hardware and must be unique across an entire internetwork.

**NOTE** *Addressing is done at the data link layer as well, but those addresses refer to local physical devices.*

**Routing** The defining function of the network layer is routing—moving data across a series of interconnected networks. It is the job of the devices and software routines that function at the network layer to handle incoming packets from various sources, determine their final destination, and then figure out where they need to be sent to get them where they are supposed to go. (You'll find a more complete discussion of routing in the OSI model in the section covering indirect device connection in Chapter 5.)

**Datagram Encapsulation** The network layer normally *encapsulates* messages received from higher layers by placing them into *datagrams* (also called *packets*) with a network layer header (the previous chapter discusses encapsulation).

**Fragmentation and Reassembly** The network layer must send messages down to the data link layer for transmission. Some data link layer technologies limit the length of any message that can be sent. If the packet that the network layer wants to send is too large, the network layer must split the packet up (fragment it), send each

piece to the data link layer, and then have the pieces reassembled once they arrive at the network layer on the destination machine. The IP is the best-known example of a protocol that performs these functions; see Chapter 22 for a discussion of IP datagram fragmentation.

**Error Handling and Diagnostics** The network layer uses special protocols to allow devices that are logically connected (or that are trying to route traffic) to exchange information about the status of hosts on the network or the devices themselves.

Network layer protocols offer either connection-oriented or connectionless services for delivering packets across the network. Connectionless ones are far more common at the network layer. In many protocol suites, the network layer protocol is connectionless, and connection-oriented services are provided by the transport layer. For example, in TCP/IP, IP is connectionless, while the layer 4 Transmission Control Protocol (TCP) is connection-oriented. Connection-oriented and connectionless protocols are discussed thoroughly in Chapter 1.

The most common network layer protocol is IP, which is why I have already mentioned it a couple of times. IP is the backbone of the Internet and the foundation of the entire TCP/IP protocol suite. There are also several protocols directly related to IP that work with it at the network layer, such as IPsec, IP NAT, and Mobile IP. The Internet Control Message Protocol (ICMP) is the main error-handling and control protocol that is used along with IP. Another notable network layer protocol outside the TCP/IP world is the Novell Internetworking Packet Exchange (IPX) protocol.

**KEY CONCEPT** The OSI Reference Model's third layer is the *network layer*. This is one of the most important layers in the model; it is responsible for the tasks that link together individual networks into *internetworks*. Network layer functions include internetwork-level addressing, routing, datagram encapsulation, fragmentation and reassembly, and certain types of error handling and diagnostics. The network layer and transport layer are closely related to each other.

The network interconnection devices that operate at the network layer are usually called *routers*. They are responsible for the routing functions I have mentioned, because they receive packets as they are sent along each “hop” of a route and send them on the next leg of their trip. They communicate with each other using routing protocols in order to determine the best routes for sending traffic efficiently. So-called brouters also reside, at least in part, at the network layer, as do the rather obviously named layer 3 switches.

## Transport Layer (Layer 4)

The fourth layer of the OSI Reference Model protocol stack is the *transport layer*, also called the *middle layer*. The transport layer is in some ways part of both the lower and upper groups of layers in the OSI model. It is more often associated with the lower layers, because it concerns itself with the *transport* of data, but its functions are also somewhat high level, resulting in its having a fair bit in common with layers 5 through 7 as well.

You'll recall that layers 1 through 3 are concerned with the actual packaging, addressing, routing, and delivery of data. The physical layer handles the bits, the data link layer deals with local networks, and the network layer handles routing between networks. The transport layer, in contrast, is sufficiently conceptual that it no longer concerns itself with these nuts-and-bolts matters. It relies on the lower layers to move data between devices.

The transport layer acts as a liaison of sorts between the abstract world of applications at the higher layers and the concrete functions of layers 1 to 3. Its overall job is to provide the necessary functions to enable communication between software application processes on different computers, which encompasses a number of different but related duties.

Because modern computers are multitasking, many different software applications may be trying to send and receive data to the same machine at any given point. The transport layer is charged with providing a means by which these applications can all send and receive data using the same lower-layer protocol implementation. Thus, it is sometimes said to be responsible for *end-to-end* or *host-to-host* transport (in fact, the equivalent layer in the TCP/IP model is called the host-to-host transport layer).

To accomplish this communication between processes, the transport layer must perform several different but related jobs. For transmission, it must track the data from each application, then combine it into a single flow of data to send to the lower layers. The device receiving information must reverse these operations, fragment the data, and funnel it to the appropriate recipient processes. The transport layer is also responsible for defining the means by which potentially large amounts of application data are fragmented for transmission.

The transport layer is also responsible for providing *connection services* for the protocols and applications that run at the levels above it. These can be categorized as either connection-oriented services or connectionless services, and each has its uses. While connection-oriented services can be handled at the network layer, they are more often seen in the transport layer in the real world. (Some protocol suites, such as TCP/IP, provide both a connection-oriented and a connectionless transport layer protocol that suits the needs of different applications.)

The transport layer is also where functions are normally included for adding features to end-to-end data transport. Whereas network layer protocols are normally concerned with just "best-effort" communications for which delivery is not guaranteed, transport layer protocols are given intelligence in the form of algorithms that ensure the reliable and efficient communication between devices. This intelligence encompasses several related jobs, including lost transmission detection and handling, and managing the rate at which data is sent in order to ensure that the receiving device is not overwhelmed.

Transmission quality—ensuring that transmissions are received as sent—is so important that some networking books define the transport layer on the basis of reliability and flow-control functions. However, not all transport layer protocols provide these services. Just as a protocol suite may have a connection-oriented and a connectionless transport layer protocol, it may also have one transport layer protocol that provides reliability and data management services, and one that

doesn't. Again, this is the case with TCP/IP: There is one main transport layer protocol, TCP, that includes reliability and flow-control features, and a second, User Datagram Protocol (UDP), that doesn't.

Let's look at the specific functions often performed at the transport layer in more detail:

**Process-Level Addressing** Addressing at the transport layer is used to differentiate between software programs. This is part of what enables many different software programs to use a network layer protocol simultaneously. The best example of transport-layer process-level addressing is the TCP and UDP port mechanism that's used in TCP/IP, which allows applications to be individually referenced on any TCP/IP device.

**Multiplexing and Demultiplexing** Using the process-level addresses, transport layer protocols on a sending device *multiplex* the data received from many application programs for transport, combining them into a single stream of data to be sent. The same protocols receive data and then *demultiplex* it from the incoming stream of datagrams, and direct each one to the appropriate recipient application processes.

**Segmentation, Packaging, and Reassembly** The transport layer segments the large amounts of data it sends over the network into smaller pieces on the source machine, and then reassembles them on the destination machine. This function is similar to the fragmentation function of the network layer. Just as the network layer fragments messages to fit the limits of the data link layer, the transport layer segments messages to suit the requirements of the underlying network layer.

**Connection Establishment, Management, and Termination** Transport layer connection-oriented protocols are responsible for the series of communications required to establish a connection, maintain it as data is sent over it, and then terminate the connection when it is no longer required.

**Acknowledgments and Retransmissions** As mentioned earlier, the transport layer is where many protocols that guarantee reliable delivery of data are implemented. This is done using a variety of techniques, most commonly by combining *acknowledgment* and *retransmission timers*. The sending device starts a timer on each occasion that data is sent; if the data is received, the recipient sends back an acknowledgment to the sender to indicate successful transmission. If no acknowledgment is returned before the timer expires, the data is retransmitted. Other algorithms and techniques are usually required to support this basic process.

**Flow Control** Transport layer protocols that offer reliable delivery also often implement *flow-control* features. These features allow one device in a communication to specify to another that it must throttle back the rate at which it is sending data. This will prevent the receiver from being bogged down with data. These features allow mismatches in speed between sender and receiver to be detected and handled.

**KEY CONCEPT** The fourth and middle OSI Reference Model layer is the *transport layer*. This layer represents the transition point between the lower layers that deal with data delivery issues, and the higher ones that work with application software. The transport layer is responsible for enabling *end-to-end communication* between application processes, which it accomplishes in part through the use of process-level addressing and multiplexing or demultiplexing. Transport layer protocols are responsible for segmenting application data into blocks for transmission and may be either connection-oriented or connectionless. Protocols at this layer also often provide data delivery management services such as reliability and flow control.

In theory, the transport and network layers are distinct, but in practice, they are often very closely related to each other. You can see this easily just by looking at the names of common protocol stacks. They are often named after the layer 3 and 4 protocols in the suite, thereby implying their close relationship. For example, the name TCP/IP comes from the suite's most commonly used transport layer protocol (TCP) and network layer protocol (IP). Similarly, the Novell NetWare suite is often called IPX/SPX for its layer 3 (IPX) and layer 4 (Sequenced Packet Exchange, or SPX) protocols.

Typically, specific transport layer protocols use the network layers in the same family. You won't often find a network using the transport layer protocol from one suite and the network layer protocol from another. The most commonly used transport layer protocols are TCP and UDP in the TCP/IP suite, SPX in the NetWare protocol suite, and NetBEUI in the NetBIOS/NetBEUI/NBF suite (though NetBEUI is more difficult to categorize).

## Session Layer (Layer 5)

The fifth layer in the OSI Reference Model is the *session layer*. As you proceed up the OSI layer stack from the bottom, the session layer is the first one where essentially all practical matters related to the addressing, packaging, and delivery of data are left behind; they are functions of layers 4 and below. The session layer is the lowest of the three upper layers, which, as a group, are concerned mainly with software application issues and not with the details of network and internetwork implementation.

The name session layer is telling: It is designed to allow devices to establish and manage *sessions*. In general terms, a session is a persistent logical linking of two software application processes that allows them to exchange data over time. In some discussions, these sessions are called *dialogs*, and, in fact, they are roughly analogous to a telephone call made between two people.

Session layer protocols primarily provide the necessary means for setting up, managing, and ending sessions. In fact, in some ways, session-layer software products resemble sets of tools more than specific protocols. These session-layer tools are normally provided to higher-layer protocols through command sets that are often called *application program interfaces* or *APIs*.

Common APIs include NetBIOS, TCP/IP Sockets, and Remote Procedure Calls (RPCs). APIs allow an application to easily accomplish certain high-level communications over the network by using a standardized set of services. Most of these session-layer tools are of primary interest to the developers of application

software. The programmers use the APIs to write software that is able to communicate using TCP/IP without developers having to know the implementation details of how TCP/IP works.

For example, the Sockets interface lies conceptually at layer 5 and is used by TCP/IP application programmers to create sessions between software programs over the Internet on the UNIX operating system. Windows Sockets similarly lets programmers create Windows software that is Internet capable and able to interact easily with other software that uses that interface. (Strictly speaking, Sockets is not a protocol, but rather a programming method.)

**NOTE** *The boundaries between layers start to blur once you get to the session layer. This makes it hard to categorize what exactly belongs at layer 5, and some technologies really span layers 5 through 7. In the world of TCP/IP in particular, it is not common to identify protocols that are specific to the OSI session layer.*

**KEY CONCEPT** The fifth layer in the OSI Reference Model layer is the *session layer*. As its name suggests, it is the layer intended to provide functions for establishing and managing sessions between software processes. Session layer technologies are often implemented as sets of software tools called *application program interfaces (APIs)*, which provide a consistent set of services that allow programmers to develop networking applications without needing to worry about lower-level details of transport, addressing, and delivery.

**NOTE** *The term “session” is somewhat vague, which means that there is sometimes disagreement on the specific functions that belong at the session layer, or about whether certain protocols belong at the session layer or not. To add to this potential confusion, there is the matter of differentiating between a connection and a session. Connections are normally the province of layer 4 and layer 3, yet a TCP connection, for example, can persist for a long time. The longevity of TCP connections makes them hard to distinguish from sessions (and there are some people who feel that the TCP/IP host-to-host transport layer really straddles OSI layers 4 and 5).*

## Presentation Layer (Layer 6)

The *presentation layer* is the sixth layer of the OSI Reference Model protocol stack and second from the top. It differs from the other layers in two key respects. First, it has a much more limited and specific function than the other layers. Second, it is used much less often than the other layers and is not required by many types of communications.

This layer deals with the *presentation* of data. More specifically, it is charged with taking care of any issues that might arise when data sent from one system needs to be viewed in a different way by the receiving system. The presentation layer also handles any special processing that must be done to data from the time an application tries to send it until the time it is sent over the network.

Here are some of the specific types of data-handling issues that the presentation layer handles:

**Translation** Many different types of computers can exist on the same network, such as PCs, Macs, UNIX systems, AS/400 servers, and mainframes. Each has many distinct characteristics and represents data in different ways (with different character sets, for example). The presentation layer hides the differences between machines.

**Compression** Compression (and decompression) may be done at the presentation layer to improve the throughput of data.

**Encryption** Some types of encryption (and decryption) are performed at the presentation layer to ensure the security of the data as it travels down the protocol stack. For example, one of the most popular encryption schemes usually associated with the presentation layer is the Secure Sockets Layer (SSL) protocol. (Some encryption is done at lower layers in the protocol stack in technologies such as IPsec.)

The presentation layer is not always used in network communications because these functions mentioned are simply not always needed. Compression and encryption are usually considered optional, and translation features are needed only in certain circumstances. Also, the presentation layer's functions may be performed at the application layer.

**NOTE** *Since its translation job isn't always needed, the presentation layer is commonly skipped by actual protocol stack implementations; in such implementations protocols at layer 7 may talk directly with those at layer 5. This is part of the reason why all of the functions at layers 5 through 7 may be included in the same software package, as described in the overview of layers and layer groupings in the previous chapter.*

**KEY CONCEPT** The sixth OSI model layer is the *presentation layer*. Protocols at this layer take care of manipulation tasks that transform data from one representation to another, such as translation, compression, and encryption. In many cases, no such functions are required in a particular networking stack; if so, there may not be any protocol active at layer 6, so layer 7 may deal with layer 5.

## Application Layer (Layer 7)

At the very top of the OSI Reference Model stack of layers, you find layer 7, the *application layer*. Continuing the trend that you saw in layers 5 and 6, this one is also named very appropriately. The application layer is the one that is used by network applications. These programs are what actually implement the functions performed by users to accomplish various tasks over the network.

It's important to understand that what the OSI model calls an application is not exactly the same as what you normally think of as an application. In the OSI model, the application layer provides services for user applications to employ.

For example, when you use your web browser, that actual software is an application running on your PC. It doesn't really reside at the application layer. Rather, it makes use of the services offered by a protocol that operates at the application layer, which is called the Hypertext Transfer Protocol (HTTP). The distinction between the browser and HTTP is subtle but important.

Not all user applications use the network's application layer in the same way. Sure, your web browser, email client, and Usenet newsreader do, but if you open a file over the network with a text editor, that editor is not using the application layer—it just sees a file addressed with a name that has been mapped to a network somewhere else. The operating system redirects what the editor does, over the network.

Similarly, not all uses of the application layer are by applications. The operating system itself can (and does) use services directly at the application layer.

That caveat aside, under normal circumstances, whenever you interact with a program on your computer that is designed specifically for use on a network, you are dealing directly with the application layer. For example, sending an email message, firing up a web browser, and using a chat program involve protocols that reside at the application layer.

**NOTE** *There are dozens of different application layer protocols. Some of the most popular ones include HTTP, FTP, SMTP, DHCP, NFS, Telnet, SNMP, POP3, NNTP, and IRC. I describe all of these and more in Section III.*

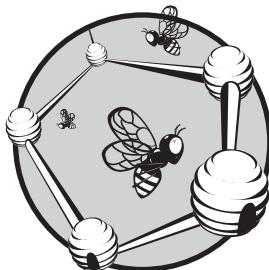
As the top-of-the-stack layer, the application layer is the only one that does not provide any services to the layer above it in the stack—there isn’t one! Instead, it provides services to programs that want to use the network, and to you, the user. So the responsibilities at this layer are simply to implement the functions that are needed by users of the network and to issue the appropriate commands to make use of the services provided by the lower layers.

**KEY CONCEPT** The *application layer* is the seventh and highest layer in the OSI Reference Model. Application protocols that implement specific user applications and other high-level functions are defined at this layer. Since they are at the top of the stack, application protocols are the only ones that do not provide services to a higher layer; they use services provided by the layers below.

As you’ve seen, the distinctions between the top three layers in the OSI Model are not very clear. In the case of TCP/IP, this is exacerbated by the decision not to separate out the session, presentation, and application layer functions. All of the protocols mentioned earlier are from the TCP/IP protocol family, and some may cover all three of the top three OSI layers, two of them, or one; in the TCP/IP model, they are all just considered applications.

# 7

## OSI REFERENCE MODEL SUMMARY



Many students of networking find the OSI Reference Model challenging to deal with. One main reason for this is that the model is somewhat *abstract*, making it hard to understand and even more difficult to apply to real networking situations. For this reason, I have included in this chapter a set of three tools that I hope will help you better understand and remember the OSI Reference Model's and concepts: an analogy, a set of mnemonics, and a summary table of OSI model layers.

### Understanding the OSI Model: An Analogy

I have attempted in this discussion of the OSI Reference Model to provide as much a plain English explanation of how it works as possible. However, there are situations in which a good analogy can accomplish what lots of descriptions cannot. So I am going to illustrate the key OSI model concepts by way of a real-life analogy. You can be the judge of whether it is a *good* analogy or not. Just remember that no analogy is perfect!

Our scenario seems relatively simple and common: The CEO of a Fortune 500 company needs to send a letter to the CEO of another company. Simple, right? Just like firing up your web browser and connecting to your favorite website is simple. However, in both cases, a lot goes on behind the scenes to make the communication happen. In the analogy shown in Table 7-1, I compare these real-world and cyber-world communications.

**Table 7-1:** OSI Reference Model Real-World Analogy

Phase	OSI Layer	CEO Letter	Website Connection (Simplified)
Transmission	7	The CEO of a company in Phoenix decides he needs to send a letter to a peer in Albany. He dictates the letter to his administrative assistant.	You decide you want to connect to the web server at IP address 10.0.12.34, which is within your organization but not on your local network. You type the address into your browser.
	6	The administrative assistant transcribes the dictation into writing.	With a website connection, nothing usually happens here. Format translation may be done in some cases.
	5	The administrative assistant puts the letter in an envelope and gives it to the mail room. The assistant doesn't actually know how the letter will be sent, but knows it is urgent, so he says, "Get this to its destination quickly."	The request is sent via a call to an API, which issues the command necessary to contact the server at that address.
	4	The mail room must decide how to get the letter where it needs to go. Since it is a rush, the people in the mail room decide to give the envelope to a courier company to send.	TCP is used to create a segment that will be sent to IP address 10.0.12.34.
	3	The courier company receives the envelope, but it needs to add its own handling information, so it places the smaller envelope in a courier envelope (encapsulation). The courier then consults its airplane route information and determines that to get this envelope to Albany, it must be flown through its hub in Chicago. It hands this envelope to the workers who load packages on its planes.	Your computer creates an IP datagram encapsulating the TCP datagram created earlier. It then addresses the packet to 10.0.12.34, but discovers that it is not on its local network. Instead, it realizes it needs to send the message to its designated routing device at IP address 10.0.43.21. It hands the packet to the driver for your Ethernet card (the software that interfaces to the Ethernet hardware).
	2	The workers take the courier envelope and put a tag on it with the code for Chicago. They then put it in a handling box and load it on the plane to Chicago.	The Ethernet card driver forms a frame containing the IP datagram and prepares it to be sent over the network. It packages the message and puts the address 10.0.43.21 (for the router) in the frame.
Routing	1	The plane flies to Chicago.	The frame is sent over the twisted-pair cable that connects your local area network. (I'm ignoring overhead, collisions, and so on, here, but then I also ignored the possibility of collisions with the plane.)

(continued)

**Table 7-1: OSI Reference Model Real-World Analogy (continued)**

Phase	OSI Layer	CEO Letter	Website Connection (Simplified)
Routing	2	In Chicago, the box is unloaded, and the courier envelope is removed from it and given to the people who handle routing in Chicago.	The Ethernet card at the machine with IP address 10.0.43.21 receives the frame, strips off the frame headers, and hands it up to the network layer.
	3	The tag marked "Chicago" is removed from the outside of the courier envelope. The envelope is then given back to the airplane workers to be sent to Albany.	The IP datagram is processed by the router, which realizes the destination (10.0.12.34) can be reached directly. It passes the datagram back down to the Ethernet driver.
	2	The envelope is given a new tag with the code for Albany, placed in another box, and loaded on the plane to Albany.	The Ethernet driver creates a new frame and prepares to send it to the device that uses IP address 10.0.12.34.
	1	The plane flies to Albany.	The frame is sent over the network.
	2	The box is unloaded, and the courier envelope is removed from the box. It is given to the Albany routing office.	The Ethernet card at the device with IP address 10.0.12.34 receives the frame, strips off the headers, and passes it up the stack.
Reception	3	The courier company in Albany sees that the destination is in Albany and delivers the envelope to the destination CEO's company.	The IP headers are removed from the datagram, and the TCP segments are handed up to TCP.
	4	The mail room removes the inner envelope from the courier envelope and delivers it to the destination CEO's assistant.	TCP removes its headers and hands the data up to the drivers on the destination machine.
	5	The assistant takes the letter out of the envelope.	The request is sent to the web-server software for processing.
	6	The assistant reads the letter and decides whether to give the letter to the CEO, transcribe it to email, call the CEO on her cell phone, or whatever.	Again, in this example nothing probably happens at the presentation layer.
	7	The second CEO receives the message that was sent by the first one.	The web server receives and processes the request.

As you can see, the processes have a fair bit in common. The vertical communication and encapsulation are pretty obvious, as is the routing. Also implied is the horizontal communication that occurs logically—the two CEOs seem to be “connected” despite all that happens to enable this to occur. Similarly, in a way, the two assistants are logically connected as well, even though they never actually converse. Of course, this example is highly simplified in just about every way imaginable, so please don’t use it as a way of trying to learn about how TCP/IP works—or courier services, for that matter!

## Remembering the OSI Model Layers: Some Mnemonics

If you spend any amount of time at all dealing with networking design or implementation issues, or learning about how the various protocols operate, the names and numbers of the various layers will eventually become second nature.

Many people, however, especially those just learning about networks, find it difficult to recall the names of all the layers, and especially, their exact order. For these people, a number of mnemonics have been created as memory aids. You probably remember mnemonics from elementary school. These are cute phrases in which each word starts with the first letter of an OSI model layer, arranged in the correct order. Some of these go in ascending layer number order, and some go in the other direction.

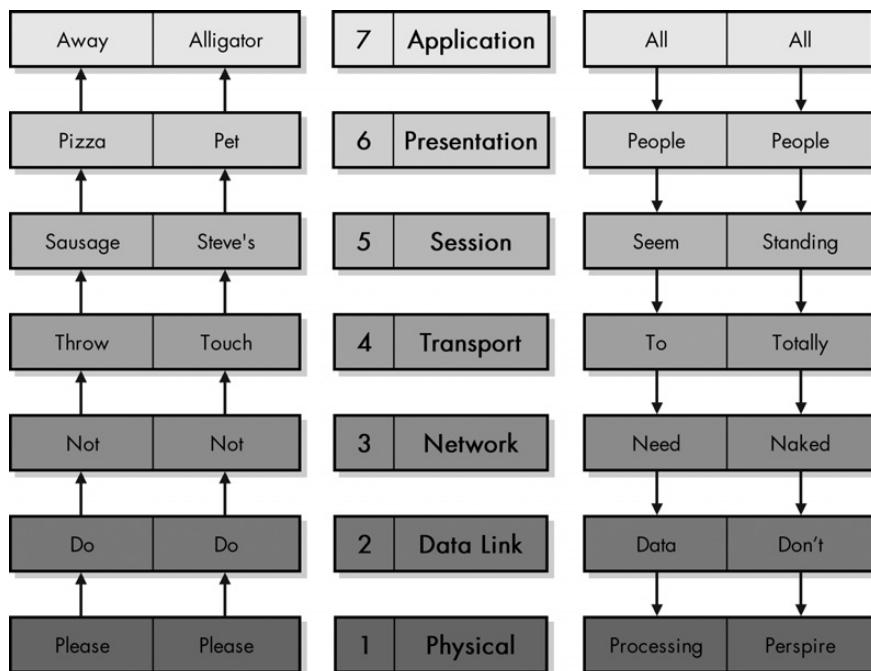
These two go from physical layer to application layer:

- Please Do Not Throw Sausage Pizza Away
- Please Do Not Touch Steve's Pet Alligator

And these go the other direction, from application to physical:

- All People Seem To Need Data Processing (a popular one)
- All People Standing Totally Naked Don't Perspire (hmm, that's interesting!)

For your convenience, I have illustrated all four of these in Figure 7-1.



**Figure 7-1: OSI Reference Model mnemonics** These mnemonics may help you to remember the order of the OSI Reference Model layers.

Or try my own creation: All People Should Teach Networking Daily Please.

## Summarizing the OSI Model Layers: A Summary Chart

To assist you in quickly comparing the layers of the OSI Reference Model, and understanding where they are different and how they relate to each other, I'm offering you the summary chart shown in Table 7-2. It shows each layer's name and number, describes its key responsibilities, talks about what type of data is generally handled at each layer, and also describes the scope of each layer in approximate terms. I also show some of the more common protocols that are associated with each layer.

The standard disclaimers still apply to this table. Namely, the layers aren't always hard-fast; I haven't listed every single protocol here, so some may really fit into more than one layer, and so on. In particular, note that many of the technologies listed as being in the data link layer are there because that is the layer where their primary functionality resides. In reality, most of these technologies include components in other layers, especially the physical layer.

**Table 7-2:** OSI Reference Model Layer Summary

Group	#	Layer Name	Key Responsibilities	Data Type Handled	Scope	Common Protocols and Technologies
Lower Layers	1	Physical	Encoding and signaling; physical data transmission; hardware specifications; topology and design	Bits	Electrical or light signals sent between local devices	Physical layers of most of the technologies listed for the data link layer
	2	Data Link	Logical link control; media access control; data framing; addressing; error detection and handling; defining requirements of physical layer	Frames	Low-level data messages between local devices	IEEE 802.2 LLC; Ethernet family; Token Ring; FDDI and CDDI; IEEE 802.11 (WLAN, Wi-Fi); HomePNA; HomeRF; ATM; SLIP and PPP
	3	Network	Logical addressing; routing; datagram encapsulation; fragmentation and reassembly; error handling and diagnostics	Datagrams/packets	Messages between local or remote devices	IP; IPv6; IP NAT; IPsec; Mobile IP; ICMP; IPX; DLC; PLP; routing protocols such as RIP and BGP
	4	Transport	Process-level addressing; multiplexing/demultiplexing; connections; segmentation and reassembly; acknowledgments and retransmissions; flow control	Datagrams/segments	Communication between software processes	TCP and UDP; SPX; NetBEUI/NBF

(continued)

**Table 7-2:** OSI Reference Model Layer Summary (continued)

<b>Group</b>	<b>#</b>	<b>Layer Name</b>	<b>Key Responsibilities</b>	<b>Data Type Handled</b>	<b>Scope</b>	<b>Common Protocols and Technologies</b>
Upper Layers	5	Session	Session establishment, management, and termination	Sessions	Sessions between local or remote devices	NetBIOS, Sockets, named pipes, RPC
	6	Presentation	Data translation; compression and encryption	Encoded user data	Application data representations	SSL; shells and redirectors; MIME
	7	Application	User application services	User data	Application data	DNS; NFS; BOOTP; DHCP; SNMP; RMON; FTP; TFTP; SMTP; POP3; IMAP; NNTP; HTTP; Telnet

# **PART I-3**

## **TCP/IP PROTOCOL SUITE AND ARCHITECTURE**

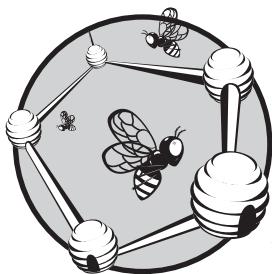
In the first two parts of this “TCP/IP Overview and Background Information” section, I have laid the groundwork for understanding how networks function in general terms. Now we can begin to turn our attention to the main subject of this book: TCP/IP. Just as Ethernet rules the roost when it comes to local area network (LAN) technologies, and IEEE 802.11 is the boss of the wireless LAN (WLAN) world, TCP/IP dominates and even defines the world of modern internetworking, including the Internet.

Since TCP/IP is the subject of this entire book, you might be wondering why this part is so small, containing only a single chapter. The reason is that it provides only a high-level overview of the TCP/IP protocol suite. TCP/IP is a collection of several dozen constituent protocols and technologies. These are described in the following two sections of the book, which cover lower-layer and application protocols, respectively. These protocols are summarized in the TCP/IP introduction chapter that follows, which also provides a brief history of TCP/IP and describes its services and model.



# 8

## **TCP/IP PROTOCOL SUITE AND ARCHITECTURE**



Named for two of its key protocols, the TCP/IP protocol suite has been in continual use and development for about three decades. In that time, it has evolved from an experimental technology that was used to hook together a handful of research computers to the powerhouse of the largest and most complex computer network in history: the global Internet, connecting together millions of networks and end devices.

In this chapter, we begin a magical tour through the mystical world of TCP/IP with an overview and a brief look at its very interesting history. I discuss the services provided in TCP/IP networks and then explain the architectural model used under TCP/IP. I then provide a brief description of each of the most important TCP/IP protocols that are discussed in the remainder of the book.

## TCP/IP Overview and History

The best place to begin an examination of TCP/IP is probably with the name itself. In fact, TCP/IP consists of dozens of different protocols, of which two are usually considered the most important. The *Internet Protocol (IP)* is the primary OSI model network layer (layer 3) protocol that provides addressing, datagram routing, and other functions in an internetwork. The *Transmission Control Protocol (TCP)* is the primary transport layer (layer 4) protocol and is responsible for connection establishment and management, and reliable data transport between software processes on devices. Because these two protocols are so important, their abbreviations have come to represent the entire suite: TCP/IP.

IP and TCP are important because many of TCP/IP's most critical functions are implemented at layers 3 and 4, where these protocols live. However, there is much more to TCP/IP than just TCP and IP. The protocol suite as a whole requires the work of many different protocols and technologies to make a functional network that can properly provide users with the applications they need.

TCP/IP uses its own four-layer architecture (which corresponds roughly to the OSI Reference Model), to provide a framework for the various protocols that compose it. It also includes numerous high-level applications, some of which are well known by Internet users who may not realize they are part of TCP/IP, such as the Hypertext Transfer Protocol (HTTP, which powers the World Wide Web) and File Transfer Protocol (FTP). In the coming discussions on TCP/IP architecture and protocols, we'll look at most of the important TCP/IP protocols and how they fit together.

### **TCP/IP History and Development**

The history of the Internet and the history of TCP/IP are so closely related that it is difficult to discuss one without also talking about the other. They were developed together, with TCP/IP providing the mechanism for implementing the Internet. Over the years, TCP/IP has continued to evolve to meet the needs of the Internet and also smaller, private networks that use the technology. We'll take a brief look at that history here.

The TCP/IP protocols were initially created as part of the research network developed by the United States *Defense Advanced Research Projects Agency (DARPA or ARPA)*. Initially, this fledgling network, called the *ARPAnet*, was designed to use a number of protocols that had been adapted from existing technologies. However, they all had flaws or limitations either in concept or in practical matters, such as capacity when used on the ARPAnet. The developers of the new network recognized that trying to use these existing protocols might eventually lead to problems as the ARPAnet increased in size and was adapted for newer uses and applications.

In 1973, the development of a full-fledged system of internetworking protocols for the ARPAnet began. Interestingly, early versions of this technology included only one core protocol: TCP. And in fact, these letters didn't even stand for what they do today; they stood for the *Transmission Control Program*. The first version of this predecessor of modern TCP was written in 1973, then revised and formally documented in RFC 675, Specification of Internet Transmission Control Program, published in December 1974.

**NOTE** Internet standards are defined in documents called Requests for Comments (RFCs). These documents, and the process used to create them, are described in Chapter 3.

Testing and development of TCP continued for several years. In March 1977, version 2 of TCP was documented. In August 1977, a significant turning point came in TCP/IP's development. Jon Postel, one of the most important pioneers of the Internet and TCP/IP, published a set of comments on the state of TCP. In that document (known as *Internet Engineering Note number 2*, or *IEN 2*), he provided an excellent example of how reference models and layers aren't just for textbooks:

We are screwing up in our design of internet protocols by violating the principle of layering. Specifically we are trying to use TCP to do two things: serve as a host level end to end protocol, and to serve as an internet packaging and routing protocol. These two things should be provided in a layered and modular way. I suggest that a new distinct internetwork protocol is needed, and that TCP be used strictly as a host level end to end protocol.

—Jon Postel, IEN 2, 1977

Postel was essentially saying that the version of TCP created in the mid-1970s was trying to do too much. Specifically, it was encompassing both OSI layer 3 and layer 4 activities. His vision was prophetic, because we now know that having TCP handle all of these activities would have indeed led to problems down the road.

Postel's observation led to the definition of TCP/IP architecture, and the splitting of TCP into TCP at the transport layer and IP at the network layer. The process of dividing TCP into two portions began in 1978 with version 3 of TCP. The first formal standard for the versions of IP and TCP used in modern networks (version 4) was created in 1980.

TCP/IP quickly became the standard protocol set for running the ARPAnet. In the 1980s, more and more machines and networks were connected to the evolving ARPAnet using TCP/IP protocols, and the TCP/IP Internet was born.

**KEY CONCEPT** TCP/IP was initially developed in the 1970s as part of an effort to define a set of technologies to operate the fledgling Internet. The name TCP/IP came about when the original *Transmission Control Program (TCP)* was split into the *Transmission Control Protocol (TCP)* and *Internet Protocol (IP)*. The first modern versions of these two key protocols were documented in 1980 as TCP version 4 and IP version 4, respectively.

### ***Important Factors in the Success of TCP/IP***

TCP/IP was at one time just one of many different sets of protocols that could be used to provide network-layer and transport-layer functionality. Today there are still other options for internetworking protocols, such as Novell's IPX/SPX, but TCP/IP is the universally accepted worldwide standard.

TCP/IP's growth in popularity has been due to a number of important factors. Some of these are historical, such as the fact that it is tied to the Internet as described earlier, while others are related to the characteristics of the protocol suite itself.

**Integrated Addressing System** TCP/IP includes within it (as part of IP primarily) a system for identifying and addressing devices on both small and large networks. The addressing system is designed to allow devices to be addressed regardless of the lower-level details of how each constituent network is constructed. Over time, the mechanisms for addressing in TCP/IP have improved to meet the needs of growing networks, especially the Internet. The addressing system also includes a centralized administration capability for the Internet to ensure that each device has a unique address.

**Design for Routing** TCP/IP is specifically designed to facilitate the routing of information over a network of arbitrary complexity. In fact, TCP/IP is conceptually concerned more with connecting networks than with connecting devices. TCP/IP routers enable data to be delivered between devices on different networks by moving it one step at a time from one network to the next. A number of support protocols in TCP/IP are designed to allow routers to exchange critical information and manage the efficient flow of information from one network to another.

**Underlying Network Independence** TCP/IP operates primarily at layers 3 and above, and includes provisions to allow it to function on almost any lower-layer technology, including local area networks (LANs), wireless LANs, and wide area networks (WANs) of various sorts. This flexibility means that you can mix and match a variety of different underlying networks and connect them all using TCP/IP.

**Scalability** One of the most amazing characteristics of TCP/IP is the scalability of its protocols. Over the decades, it has proven its mettle as the Internet has grown from a small network with just a few machines to a huge internetwork with millions of hosts. While some changes have been required periodically to support this growth, these changes have taken place as part of the TCP/IP development process, yet the core of TCP/IP is basically the same as it was 25 years ago.

**Open Standards and Development Process** The TCP/IP standards, rather than being proprietary, are open ones, freely available to the public. Furthermore, the process used to develop the TCP/IP standards is also completely open. The TCP/IP standards and protocols are developed and modified using the unique, democratic Request for Comments (RFC) process (described in Chapter 3), with all interested parties invited to participate. This ensures that anyone with an interest in the TCP/IP protocols is given a chance to provide input into their development and also ensures the worldwide acceptance of the protocol suite.

**Universality** Everyone uses TCP/IP because everyone uses it! This last point is, perhaps ironically, arguably the most important. Not only is TCP/IP the underlying language of the Internet, it is also used in most private networks today. Even former competitors to TCP/IP, such as Novell's NetWare, now use TCP/IP to carry traffic.

**KEY CONCEPT** While TCP/IP is not the only internetworking protocol suite, it is definitely the most important one. Its unparalleled success is due to a wide variety of factors. These include its technical features, such as its routing-friendly design and scalability, its historical role as the protocol suite of the Internet, and its open standards and development process, which reduce barriers to acceptance of TCP/IP protocols.

## TCP/IP Services

TCP/IP is most often studied in terms of its layer-based architecture and the protocols that it provides at those different layers. These protocols, however, represent the technical details of *how* TCP/IP works. They are of interest to us as students of technology, but are normally hidden from users who do not need to see the guts of TCP/IP to know that it works. Before proceeding to these details, let's take a bigger picture look at *what* TCP/IP does.

In the discussion of the OSI Reference Model concepts (in Chapter 5), I mentioned that the theoretical operation of the model is based on the idea of one layer providing services to the layers above it. TCP/IP covers many layers of the OSI model, and so it collectively provides services of this sort as well in many ways. Conceptually, we can divide TCP/IP services into two groups:

**Services Provided to Other Protocols** The first group of services consists of the core functions implemented by the main TCP/IP protocols such as IP, TCP, and User Datagram Protocol (UDP). These services are designed to actually accomplish the internetworking functions of the protocol suite. For example, at the network layer, IP provides functions such as addressing, delivery, datagram packaging, fragmentation, and reassembly. At the transport layer, TCP and UDP are concerned with encapsulating user data and managing connections between devices. Other protocols provide routing and management functionality. Higher-layer protocols use these services, allowing them to concentrate on what they are intended to accomplish.

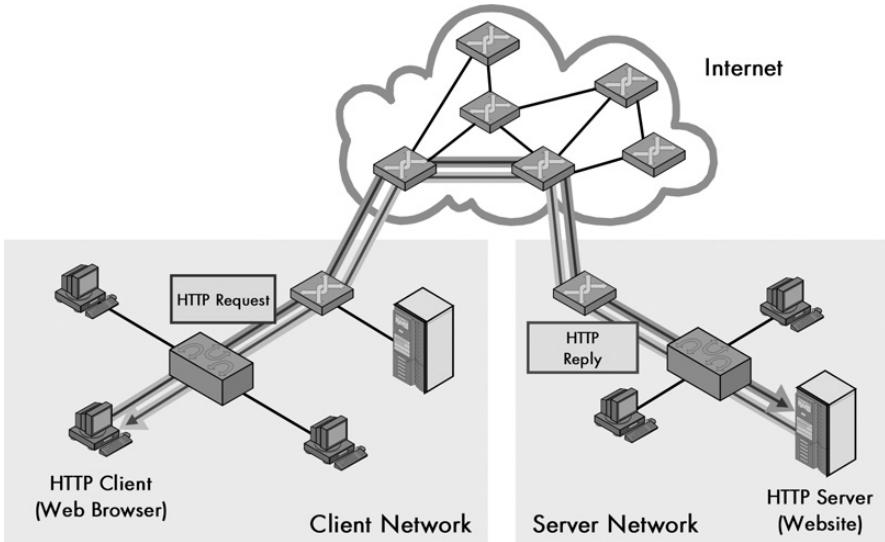
**End-User Services** The second group of services provided by TCP/IP is the set of end-user services. These facilitate the operation of the applications that users run to make use of the power of the Internet and other TCP/IP networks. For example, the Web is arguably the most important Internet application. Web services are provided through HTTP, a TCP/IP application layer protocol. HTTP in turn uses services provided by lower-level protocols. All of these details are hidden from end users, entirely on purpose.

## The TCP/IP Client/Server Structural Model

TCP/IP services primarily operate in the *client/server* model. It's a system in which a relatively small number of server machines provides services to a much larger number of client hosts (see Chapter 1).

Just as client/server networking applies to hardware, this same concept underlies the design of the TCP/IP protocols and software applications, as shown in Figure 8-1.

TCP/IP protocols are not set up so that two machines that want to communicate use identical software. Instead, a conscious decision was made to make communication function using matched, complementary pairs of client and server software. The client initiates communication by sending a request to a server for data or other information. The server then responds with a reply to the client, giving the client what it requested, or else it replies with an alternative response, such as an error message or information about where else it might find the data. Most (but not all) TCP/IP functions work in this manner.



**Figure 8-1: TCP/IP client/server operation** Most TCP/IP protocols involve communication between two devices, typically as client and server, such as this web (HTTP) transaction over the Internet.

Figure 8-1 is a simplified illustration that shows a common example—a web transaction using HTTP. The web browser is an HTTP client and initiates the communication with a request for a file or other resource sent over the Internet to a website, which is an HTTP server. The server responds to the client with the information requested. (Servers generally respond to many clients simultaneously.)

There are numerous advantages to client/server operation in TCP/IP. Just as client hardware and server hardware can be tailored to their very different jobs, client software and server software can also be optimized to perform their jobs as efficiently as possible. For example, to get information from the Web, web-client software (usually called a *browser*) sends requests to a web server. The web server then responds with the requested content. (There's more to it than that, of course, but that's how it appears to the user.) Among other things, the web browser allows the user to communicate with web servers; the web-server software is designed to receive and respond to requests.

The terms *client* and *server* can be confusing in TCP/IP because they are used in several different ways, sometimes simultaneously.

**KEY CONCEPT** The TCP/IP protocol suite is oriented around the notion of *client/server* network communication. Rather than all devices and protocol software elements being designed as peers, they are constructed as matched sets. Clients normally initiate communications by sending requests, and servers respond to such requests, providing the client with the desired data or an informative reply.

## **Hardware and Software Roles**

The terms *client* and *server* usually refer to the primary roles played by networked hardware. A *client* computer is usually something like a PC or Macintosh used by an individual; it primarily initiates conversations by sending requests. A *server* is usually a high-powered machine dedicated to responding to client requests, sitting in a computer room somewhere that no one but its administrator ever sees.

As mentioned earlier, TCP/IP uses different software for many protocols to implement client and server roles. For example, a web browser is a piece of client software, while web-server software is completely different. Client software is usually found on client hardware and server software on server hardware, but some devices may run both client and server software.

## **Transactional Roles**

In any exchange of information, the client is normally the device that initiates communication or sends a query, and then the server responds, usually by providing information. Again, the client software on a client device usually initiates the transaction.

In a typical organization there will be many smaller individual computers designated as clients and a few larger ones that are servers. The servers normally run server software, and the clients run client software. But servers can also be set up with client software, and clients can be set up with server software.

For example, suppose you are an administrator working in the computer room on server 1 and need to transfer a file to server 2. You start an FTP session to initiate the file transfer with server 2. In this transaction, server 1 is the client, since it is initiating communication. Theoretically, you could even start an FTP transfer from server 1 to a particular client, if that client had FTP server software to answer the server's request. (This is less common, because server software is often not installed on client machines.)

Transactional roles come into play when communication occurs between servers in certain protocols. For example, when two Simple Mail Transfer Protocol (SMTP) servers communicate to exchange email (even though they are both server programs running on server hardware), during any transaction, one device acts as the client while the other acts as the server. In some cases, devices can even swap client and server roles in the middle of a session.

**KEY CONCEPT** Understanding client/server computing concepts in TCP/IP is made more complex due to the very different meanings that the terms client and server can have in various contexts. The two terms can refer to *hardware roles*—designations given to hardware devices based on whether they usually function as clients or as servers. The terms can also refer to *software roles*, meaning whether protocol software components function as clients or servers. And they can refer to *transactional roles*, meaning whether a device and program functions as a client or server in any given exchange of data.

**NOTE** The client and server roles I have discussed are the traditional ones. The rise of powerful personal computers and widespread Internet access (especially always-on broadband connectivity) has led to a significant blurring of client and server hardware and software. Many client machines now include server software that allows them to, for example, respond to World Wide Web queries from other clients. Also, many file-sharing programs allow clients to communicate using the peer-to-peer structural model. However, most TCP/IP communication is still client/server in nature, so it's important to keep these roles in mind.

## TCP/IP Architecture and the TCP/IP Model

The OSI Reference Model's seven layers divide up the tasks required to implement a network, as described in Part I-2 of this book. However, it is not the only such model. In fact, the TCP/IP protocol suite was developed before the OSI Reference Model; as such, its inventors didn't use the OSI model to explain TCP/IP architecture (even though the OSI model is often used in TCP/IP discussions today). The developers of TCP/IP created their own architectural model, which goes by different names including the *TCP/IP model*, the *DARPA model* (after the agency that was largely responsible for developing TCP/IP) and the *DoD model* (after the United States Department of Defense). Most people call it the TCP/IP model.

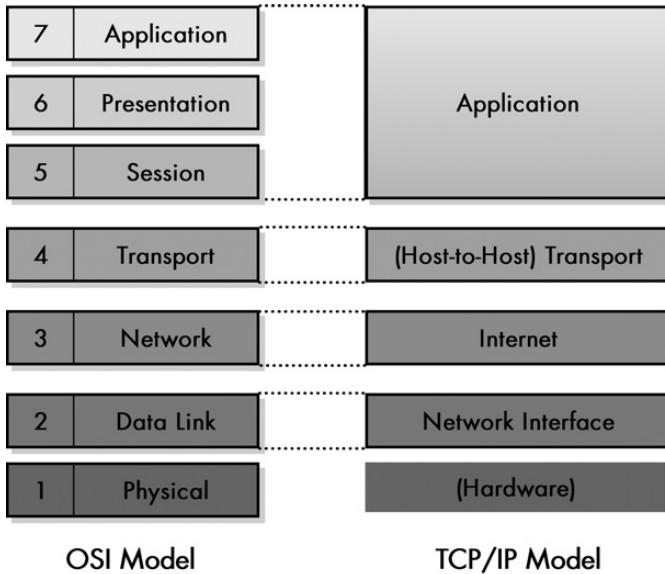
Regardless of the model you use to represent the function of a network, the model's functions are pretty much the same. The TCP/IP and the OSI models are really quite similar, even if they don't carve up the network functionality precisely the same way.

Since the OSI model is so widely used, it is common to explain the TCP/IP architecture both in terms of the TCP/IP layers and the corresponding OSI layers. Figure 8-2 shows the relationship between the two models. The TCP/IP model does not address the physical layer, where hardware devices reside. The next three layers—*network interface*, *internet*, and *host-to-host transport*—correspond to layers 2, 3, and 4 of the OSI model. The TCP/IP *application* layer conceptually blurs the top three OSI layers. Note, too, that some people consider certain aspects of the OSI session layer to be part of the TCP/IP host-to-host transport layer.

As shown in Figure 8-2, the TCP/IP model uses four layers that logically span the equivalent of the top six layers of the OSI model. (The physical layer is not covered by the TCP/IP model because the data link layer is considered the point at which the interface occurs between the TCP/IP stack and the underlying networking hardware.) Starting from the bottom, the TCP/IP layers are described in the following sections.

### **Network Interface Layer**

As its name suggests, the network interface layer is where the actual TCP/IP protocols running at higher layers interface to the local network. This layer is somewhat controversial in that some people don't even consider it a legitimate part of TCP/IP, usually because none of the core IP protocols run at this layer. Despite this, the network interface layer is part of the architecture. It is equivalent to the data link layer (layer 2) in the OSI Reference Model (see Chapter 6) and is also sometimes called the *link layer*. You may also see the name *network access layer* used.



**Figure 8-2: OSI Reference Model and TCP/IP model layers** The TCP/IP architectural model has four layers that approximately match six of the seven layers in the OSI Reference Model.

On many TCP/IP networks, there is no TCP/IP protocol running at all on this layer, because it is simply not needed. For example, if you run TCP/IP over Ethernet, then Ethernet handles layer 2 (and layer 1) functions. However, the TCP/IP standards do define protocols for TCP/IP networks that do not have their own layer 2 implementation. These protocols, the Serial Line Internet Protocol (SLIP) and the Point-to-Point Protocol (PPP), fill the gap between the network layer and the physical layer. They are commonly used to facilitate TCP/IP over direct serial line connections (such as dial-up telephone networking) and other technologies that operate directly at the physical layer.

### Internet Layer

The Internet layer corresponds to the network layer in the OSI Reference Model (thus it is sometimes called the *network layer* even in TCP/IP model discussions). It is responsible for typical layer 3 jobs, such as logical device addressing, data packaging, manipulation and delivery, and routing. At this layer, you find IP, which is arguably the heart of TCP/IP, as well as support protocols such as the Internet Control Message Protocol (ICMP) and the routing protocols (RIP, OSPF, BGP, and so on). IP version 6, the next-generation IP, is also at this layer.

## **Host-to-Host Transport Layer**

This primary job of the host-to-host transport layer is to facilitate end-to-end communication over an internetwork. It is in charge of allowing logical connections to be made between devices that allow data to be sent either unreliably (with no guarantee that it gets there) or reliably (where the protocol keeps track of the data sent and received in order to make sure it arrives, and resends it if necessary). It is also here that identification of the specific source and destination application process is accomplished.

The formal name of this layer is often shortened to just the *transport layer*. The key TCP/IP protocols at this layer are TCP and UDP. The TCP/IP transport layer corresponds to the layer of the same name in the OSI model (layer 4) but includes certain elements that are arguably part of the OSI session layer. For example, TCP establishes a connection that can persist for a long period of time, which some people say makes a TCP connection more like a session.

## **Application Layer**

The application layer is the highest layer in the TCP/IP model. It is a rather broad layer, encompassing layers 5 through 7 in the OSI model. While this seems to represent a loss of detail compared to the OSI model, that's probably a good thing. The TCP/IP model better reflects the somewhat fuzzy nature of the divisions between the functions of the higher layers in the OSI model, which in practical terms often seem rather arbitrary. It really is hard to separate some protocols in terms of which portions of layers 5, 6, or 7 they encompass.

Numerous protocols reside at the application layer. These include application protocols such as HTTP, FTP, and SMTP for providing end-user services as well as administrative protocols like Simple Network Management Protocol (SNMP), Dynamic Host Configuration Protocol (DHCP), and Domain Name System (DNS).

**NOTE** *The Internet and host-to-host transport layers are usually considered the core of TCP/IP architecture, because they contain most of the key protocols that implement TCP/IP internetworks.*

In the following section, I provide a brief look at each of the TCP/IP protocols covered in detail in this book and offer more detail on where they all fit into the TCP/IP architecture. I also discuss a couple of protocols that, interestingly, don't really fit into the TCP/IP layer model very well at all.

**KEY CONCEPT** The architecture of the TCP/IP protocol suite is often described in terms of a layered reference model called the *TCP/IP model*, *DARPA model*, or *DoD model*. The TCP/IP model includes four layers: the *network interface layer* (responsible for interfacing the suite to the physical hardware on which it runs), the *Internet layer* (where device addressing, basic datagram communication, and routing take place), the *host-to-host transport layer* (where connections are managed and reliable communication is ensured), and the *application layer* (where end-user applications and services reside). The first three layers correspond to layers 2 through 4 of the OSI Reference Model respectively; the application layer is equivalent to OSI layers 5 to 7.

## TCP/IP Protocols

Since TCP/IP is a protocol suite, it is most often discussed in terms of the protocols that compose it. Each protocol resides in a particular layer of the TCP/IP architectural model that I just discussed and is charged with performing a certain subset of the total functionality required to implement a TCP/IP network or application. The protocols work together to allow TCP/IP as a whole to operate.

**NOTE** You will sometimes hear TCP/IP called just a protocol instead of a protocol suite. This is a simplification that, while technically incorrect, is widely used. I believe it arises in large part due to Microsoft referring to protocol suites as protocols in its operating systems. I discuss this issue in more detail in Chapter 1.

As mentioned earlier, a few TCP/IP protocols are usually called the core of the suite, because they are responsible for its basic operation. In this core, most people would include the main protocols at the Internet and transport layers: IP, TCP, and UDP. These core protocols support many other protocols in order to perform a variety of functions at each of the TCP/IP model layers.

**NOTE** On the whole, there are many hundreds of TCP/IP protocols and applications, and I could not begin to cover each and every one in this book. I do include chapters in which I discuss several dozen of the protocols that I consider important for one reason or another. Full coverage of each of these protocols (to varying levels of detail) can be found in Section II and Section III of this book.

Table 8-1 contains a summary of each of the TCP/IP protocols discussed in this book. I have organized them by layer, and I have provided cross-references to the chapters where each is discussed. The organization of protocols in the TCP/IP protocol suite can also be seen at a glance in Figure 8-3. I have also shown in the network interface layer where TCP/IP hardware drivers conceptually reside; these are used at layer 2 when TCP/IP is implemented on a LAN or WAN technology, rather than using SLIP or PPP.

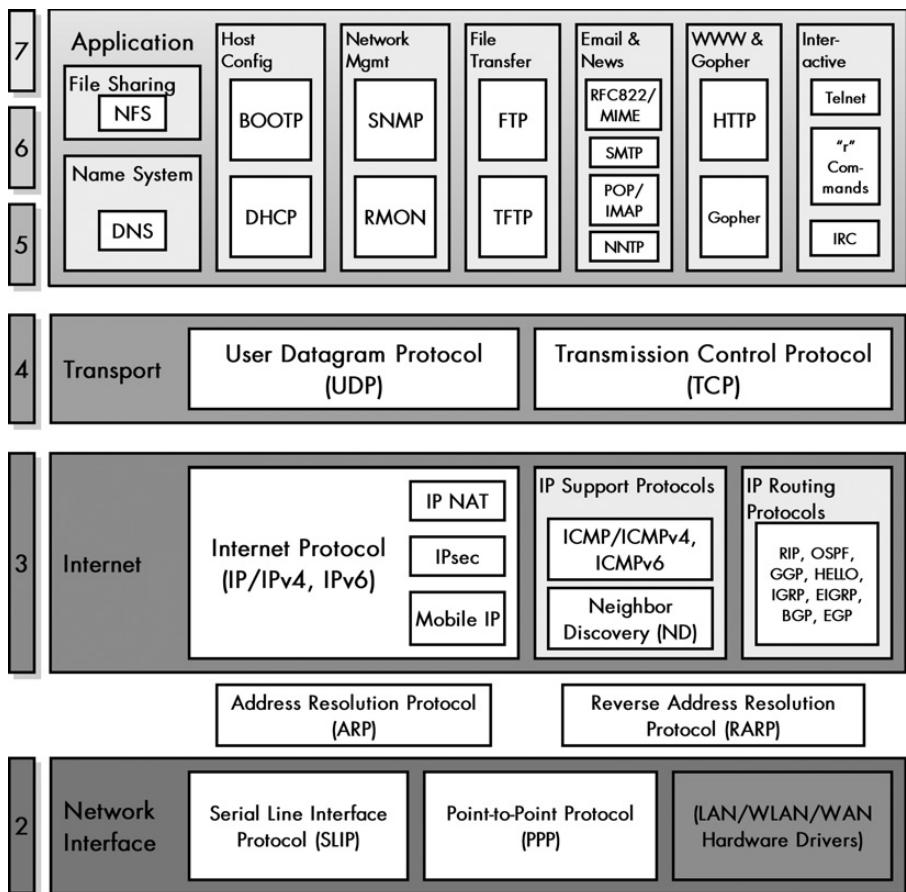
**Table 8-1: TCP/IP Protocols**

<b>TCP/IP Layer</b>	<b>Protocol Name</b>	<b>Protocol Abbr.</b>	<b>Description</b>
Network Interface (Layer 2)	Serial Line Internet Protocol	SLIP	Provides basic TCP/IP functionality by creating a layer 2 connection between two devices over a serial line. See Chapter 9.
	Point-to-Point Protocol	PPP	Provides layer 2 connectivity like SLIP, but is much more sophisticated and capable. PPP is itself a suite of protocols (subprotocols, if you will) that allow for functions such as authentication, data encapsulation, encryption, and aggregation, thereby facilitating TCP/IP operation over WAN links. See Chapters 9-12.
Network Interface/Internet (Layer 2/3)	Address Resolution Protocol	ARP	Used to map layer 3 IP addresses to layer 2 physical network addresses. See Chapter 13.
	Reverse Address Resolution Protocol	RARP	Determines the layer 3 address of a machine from its layer 2 address. Now mostly superseded by BOOTP and DHCP. See Chapter 14.
Internet Layer (Layer 3)	Internet Protocol, Internet Protocol Version 6	IP, IPv6	Provides encapsulation and connectionless delivery of transport layer messages over a TCP/IP network. Also responsible for addressing and routing functions. See Part II-3 and Part II-4.
	IP Network Address Translation	IP NAT	Allows addresses on a private network to be automatically translated to different addresses on a public network, thereby providing address sharing and security benefits. (Note that some people don't consider IP NAT to be a protocol in the strict sense of that word.) See Chapter 28.
	IP Security	IPsec	A set of IP-related protocols that improve the security of IP transmissions. See Chapter 29.
	Internet Protocol Mobility Support	Mobile IP	Resolves certain problems with IP associated with mobile devices. See Chapter 30.
	Internet Control Message Protocol	ICMP/ICMPv4, ICMPv6	A support protocol for IP and IPv6 that provides error reporting and information request-and-reply capabilities to hosts. See Part II-6.
	Neighbor Discovery Protocol	NDP	A new support protocol for IPv6 that includes several functions performed by ARP and ICMP in conventional IP. See Chapter 36.
	Routing Information Protocol, Open Shortest Path First, Gateway-to-Gateway Protocol, HELLO Protocol, Interior Gateway Routing Protocol, Enhanced Interior Gateway Routing Protocol, Border Gateway Protocol, Exterior Gateway Protocol	RIP, OSPF, GGP, HELLO, IGRP, EIGRP, BGP, EGP	Protocols used to support the routing of IP datagrams and the exchange of routing information. See Part II-7.
Host-to-Host Transport Layer (Layer 4)	Transmission Control Protocol	TCP	The main transport layer protocol for TCP/IP. Establishes and manages connections between devices and ensures reliable and flow-controlled delivery of data using IP. See Part II-8.

*(continued)*

**Table 8-1: TCP/IP Protocols (continued)**

<b>TCP/IP Layer</b>	<b>Protocol Name</b>	<b>Protocol Abbr.</b>	<b>Description</b>
Host-to-Host Transport Layer (Layer 4) <i>continued</i>	User Datagram Protocol	UDP	A transport protocol that can be considered a severely stripped-down version of TCP. It is used to send data in a simple way between application processes, without the many reliability and flow-management features of TCP, but often with greater efficiency. See Chapter 44.
Application Layer (Layer 5/6/7)	Domain Name System	DNS	Provides the ability to refer to IP devices using names instead of just numerical IP addresses. Allows machines to resolve these names into their corresponding IP addresses. See Part III-1.
	Network File System	NFS	Allows files to be shared seamlessly across TCP/IP networks. See Chapter 58.
	Bootstrap Protocol	BOOTP	Developed to address some of the issues with RARP and used in a similar manner: to allow the configuration of a TCP/IP device at startup. Generally superseded by DHCP. See Chapter 60.
	Dynamic Host Configuration Protocol	DHCP	A complete protocol for configuring TCP/IP devices and managing IP addresses. The successor to RARP and BOOTP, it includes numerous features and capabilities. See Part III-3.
	Simple Network Management Protocol	SNMP	A full-featured protocol for remote management of networks and devices. See Part III-4.
	Remote Monitoring	RMON	A diagnostic "protocol" (really a part of SNMP) used for remote monitoring of network devices. See Chapter 69.
	File Transfer Protocol, Trivial File Transfer Protocol	FTP, TFTP	Protocols designed to permit the transfer of all types of files from one device to another. See Part III-6.
	RFC 822, Multipurpose Internet Mail Extensions, Simple Mail Transfer Protocol, Post Office Protocol, Internet Message Access Protocol	RFC 822, MIME, SMTP, POP, IMAP	Protocols that define the formatting, delivery, and storage of email messages on TCP/IP networks. See Part III-7.
	Network News Transfer Protocol	NNTP	Enables the operation of the Usenet online community by transferring Usenet news messages between hosts. See Chapter 85.
	Hypertext Transfer Protocol	HTTP	Transfers hypertext documents between hosts; implements the World Wide Web. See Part III-8.
	Gopher Protocol	Gopher	An older document-retrieval protocol, now largely replaced by the World Wide Web. See Chapter 86.
	Telnet Protocol	Telnet	Allows a user on one machine to establish a remote terminal session on another. See Chapter 87.
	Berkeley "r" Commands	—	Permit commands and operations on one machine to be performed on another. See Chapter 87.
	Internet Relay Chat	IRC	Allows real-time chatting between TCP/IP users. See Chapter 87.
	Administration and Troubleshooting Utilities and Protocols	—	A collection of software tools that allows administrators to manage, configure, and troubleshoot TCP/IP internetworks. See Chapter 88.



**Figure 8-3: TCP/IP protocols** This diagram shows all the TCP/IP protocols covered in this book, arranged by TCP/IP and OSI Reference Model layer (with the exception of the administration utilities).

You can see in the previous table and figure that ARP and RARP are the oddballs. In some ways they belong in both layer 2 and layer 3, and in other ways they belong in neither. They really serve to link together the network interface layer and the Internet layer. For this reason, I believe they belong *between* these two and call them “layer connection” protocols. See Chapters 13 and 14 for more on this issue.

# **SECTION II**

## **TCP/IP LOWER-LAYER CORE PROTOCOLS**

The TCP/IP protocol suite is largely defined in terms of the protocols that constitute it, and several dozen are covered in this book. Most of the critical protocols of the suite function at the lower layers of the OSI Reference Model (covered in Part I-2): layers 2, 3 and 4, which correspond to the network interface, Internet, and transport layers in the TCP/IP model architecture (described in Part I-3). Included here are the all-important Internet Protocol (IP) at layer 3 and Transmission Control Protocol (TCP) at layer 4, which combine to give TCP/IP its name.

Due to the importance of these and other TCP/IP protocols at the lower layers, this is the largest of the three sections of this book. It contains eight parts. The first describes the two TCP/IP protocols that reside at the network interface layer (layer 2 of the OSI Reference Model): the Point-to-Point Protocol (PPP) and the Serial Line Interface Protocol (SLIP). The second part describes a couple of special protocols that reside architecturally between layers 2 and 3: the Address Resolution Protocol (ARP) and the Reverse Address Resolution Protocol (RARP). The third and fourth parts describe the IP versions 4 and 6 (IPv4 and IPv6). The fifth and sixth parts discuss IP-related feature and support protocols, and the seventh part describes IP routing protocols. Finally, the eighth part covers the two TCP/IP transport layer protocols, the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP), and related topics such as the use of TCP/IP ports.



# PART II-1

## TCP/IP NETWORK INTERFACE LAYER PROTOCOLS

The lowest layer of the OSI Reference Model is the physical layer, which is responsible for the nitty-gritty details of transmitting information from one place to another on a network. The layer just above the physical layer is the *data link layer*, called the *network interface layer*, or just the *link layer*, in the TCP/IP architectural model. Its primary jobs are to implement networks at the local level and to interface between the hardware-oriented physical layer and the more abstract, software-oriented functions of the network layer and the layers above it.

In the case of TCP/IP, the Internet Protocol (IP) is the main protocol at layer 3, and it serves as the foundation of the whole TCP/IP protocol suite. IP is designed to be layered on top of any number of layer 2 technologies. However, some types of connections do not include a layer 2 protocol over which IP can run. To enable TCP/IP to operate on these kinds of links, two special TCP/IP protocols operate at the network interface layer, connecting IP to the physical layer below.

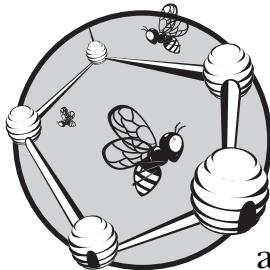
In this part, I provide a description of the two protocols that reside at the data link, or network interface layer, in the TCP/IP protocol suite. These are the older, simple Serial Line Interface Protocol (SLIP) and the newer, more

capable Point-to-Point Protocol (PPP). I begin with a chapter that provides a brief overview of SLIP and PPP, showing how they fit into the TCP/IP protocol suite as a whole and describing them in general terms.

The rest of this part contains three chapters that describe the more important of the two protocols, PPP, in more detail. The first of these three explains the core protocols that are responsible for setting up PPP links and basic operation. The second covers the protocols used to implement various special features in PPP, such as compression and encryption. The last chapter on PPP provides detailed information about the various frame formats used by PPP protocols.

# 9

## **TCP/IP SERIAL LINE INTERNET PROTOCOL (SLIP) AND POINT-TO- POINT PROTOCOL (PPP) OVERVIEW AND FUNDAMENTALS**



TCP/IP's core protocols operate at layers 3 and 4 of the OSI model, corresponding to the Internet layer and host-to-host transport layer of the TCP/IP architectural model (introduced in Chapter 8). That model also defines the network

interface layer, which corresponds to the data link layer. However, in most network implementations, TCP/IP doesn't define any protocols operating at this layer. Instead, TCP/IP assumes that layer 2 functionality is provided by a wide area network (WAN) or local area network (LAN) technology like Ethernet, Token Ring, or IEEE 802.11. These technologies are responsible for the classic layer 2 functions: physical layer addressing, media access control, and especially, layer 2 framing of datagrams received from layer 3.

There's a problem with the assumption that Internet Protocol (IP) can run on top of an existing layer 2 protocol because sometimes there isn't one. Certain technologies, such as a simple serial connection between two devices, establish only a basic, low-level connection at the physical layer.

And, of course, one type of serial connection is still *very* popular: serial dial-up networking. When you connect with a dial-up modem to your ISP, the modems negotiate a connection that architecturally exists only at the physical layer.

Since IP assumes certain services will be provided at layer 2, there is no way to make it operate directly over a serial line or other physical layer connection. At a minimum, the most important layer 2 function that is required is some mechanism for framing the IP datagram for transmission; that is, a mechanism that provides the necessary data packaging to let datagrams be transmitted over the physical layer network. Without this, IP datagrams cannot be sent over the link.

## SLIP versus PPP

To fill the gap between IP at layer 3 and the physical connection at layer 1, two protocols operate at layer 2 and provide the services that IP requires to function. One protocol is *Serial Line Internet Protocol* (SLIP), a very simple layer 2 protocol that provides only basic framing for IP. The other is *Point-to-Point Protocol* (PPP), a more complex, full-featured data link layer protocol that provides framing as well as many additional features that improve security and performance.

SLIP is extremely simple and easy to implement but lacks certain features of PPP (like authentication, compression, and error detection), which is full featured but more complicated. To draw an analogy, SLIP is a mostly sturdy, ten-year-old compact sedan, while PPP is a shiny, new luxury SUV. Both will get you from here to grandma's house, but the SUV is going to be safer, more comfortable, and better able to deal with problems that might crop up on the road. If they cost the same to buy and operate, you'd probably choose the SUV. Both SLIP and PPP cost about the same, and unlike an SUV, PPP causes no air pollution and doesn't guzzle gas. For this reason, PPP is the choice of most serial line connections today and has all but replaced SLIP.

**KEY CONCEPT** SLIP and PPP provide layer 2 connectivity for TCP/IP implementations that run directly over a physical layer link without a layer 2 technology. While SLIP is simpler, PPP is favored due to its many features and capabilities.

Both SLIP and PPP are designed for connections between just two devices; thus, the name point-to-point protocol. Since there are only two devices, A and B, communication is straightforward: A sends to B and B sends to A, and since both deal only with simple two-device connections, they do not have to manage complexities like media access control, collisions, and unique addressing schemes in the way that technologies like Ethernet must.

**NOTE** Some people don't consider SLIP and PPP to be part of the true TCP/IP protocol suite. They argue that TCP/IP is defined at layers 3 and higher on the OSI model, that IP is the basis of TCP/IP at layer 3, and that SLIP and PPP are just extra protocols that can be used under TCP/IP. To support their argument, they note that PPP can be used for protocols other than IP, which is true.

## Serial Line Internet Protocol (SLIP)

The need for a data link layer protocol to allow IP to operate over serial links was identified very early on in the development of TCP/IP. Engineers working on IP needed a way to send IP datagrams over serial links. To solve the problem, they created the very simple protocol SLIP to frame IP messages for transmission across the serial line.

Unlike most TCP/IP protocols, SLIP has never been defined as a formalized standard. It was created informally in the early 1980s, and it became the de facto standard before it was ever described in a Request for Comment (RFC). When it was published in 1988 (RFC 1055, “A Nonstandard for Transmission of IP Data-grams over Serial Lines: SLIP”), the decision was made to designate it a “nonstandard protocol.”

SLIP was designated nonstandard because it was developed as a very rudimentary, stopgap measure to provide layer 2 framing when needed. SLIP is so simple that there really isn’t much to standardize. Too, it has so many deficiencies that the Internet Engineering Task Force (IETF) apparently didn’t want to formalize it as a standard. RFC 1055 specifically mentions various problems with SLIP (as I’ll discuss later in this chapter) and the fact that work was already under way to define PPP as a more capable successor to SLIP.

**KEY CONCEPT** SLIP provides a *layer 2 framing* service for IP datagrams but no other features or capabilities.

### SLIP Data Framing Method and General Operation

SLIP performs only one function: the framing of data for transmission. Here’s how SLIP framing works. An IP datagram is passed down to SLIP, which breaks it into bytes and sends those bytes one at a time over the link. After the last byte of the datagram is sent, a special byte value is sent that tells the receiving device that the datagram has ended. This is called the SLIP *END character*, and it has a byte value of 192 in decimal numbers (C0 in hexadecimal and 11000000 binary). That’s basically SLIP framing in a nutshell: Take the whole datagram, send it one byte at a time, and then send the byte 192 to delimit the end of the datagram.

One minor enhancement to SLIP’s basic operation is to *precede* the datagram with an *END* character as well, thus clearly separating the start of the datagram from anything that precedes it. To see why this might be needed, you can imagine that at a particular time you have only one datagram to send: datagram 1. You send 1, and then send the *END* character to delimit it. Now, suppose there is a pause before the next datagram shows up. During that time, you aren’t transmitting, but if there is line noise, the other device might pick up spurious bytes here and there. If you later receive datagram 2 and just start sending it, the receiving device might think the noise bytes were part of datagram 2.

Starting datagram 2 off with an *END* character tells the recipient that anything received between this *END* character and the previous one is a separate datagram. If that’s just noise, then this “noise datagram” is just gibberish that will be rejected at the IP layer. Meanwhile, it doesn’t corrupt the real datagram you wish to send. If

no noise occurred on the line between datagrams, then the recipient will just see the *END* at the start of datagram 2 right after the one at the end of datagram 1 and will ignore the “null datagram” between the two.

But what if the *END* character is 192 in decimal numbers; what happens if the byte value 192 appears in the datagram itself? Transmitting it as is would fool the recipient into thinking that the datagram ended prematurely. To avoid this, an *Escape character (ESC)* is defined, which has a decimal value of 219 (DB in hex, 11011011 in binary). This symbol means that “this byte and the next are special.” When a value of 192 appears in the datagram, the sending device replaces it with the ESC character followed by the value 220 decimal. Thus, a single 192 becomes 219 220 (or DB DC in hexadecimal). The recipient translates back from 219 220 to 192.

**NOTE** *The SLIP ESC character is not the same as the ASCII ESC character. They both perform an “escaping” operation but are otherwise unrelated. If the ESC character itself is in the original datagram—that is, if there’s a byte value of 219 in the IP datagram to be sent—the device uses 219 221 instead of just 219.*

To summarize, SLIP does the following:

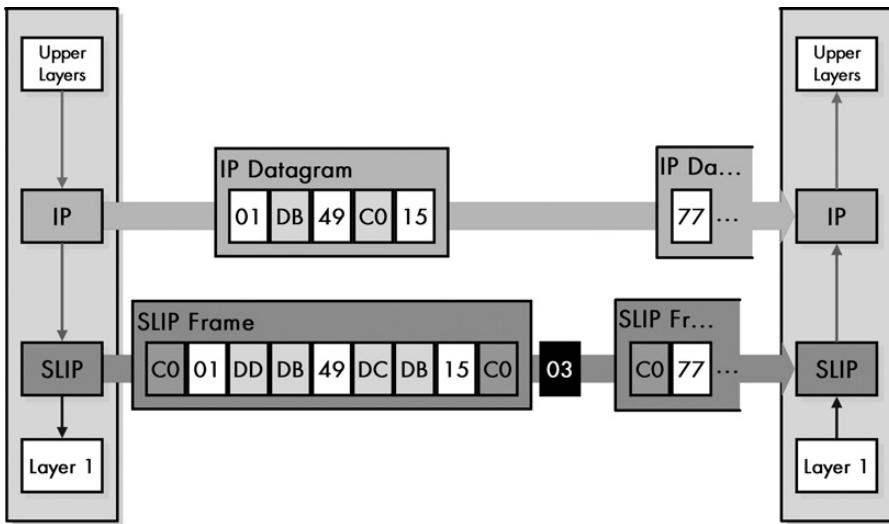
- Breaks an IP datagram into bytes
- Sends the *END* character (value 192) after the last byte of the datagram; in better implementations, it sends the *END* character before the first byte as well
- Replaces any byte to be sent in the datagram that is 192 with 219 220
- Replaces any byte to be sent that is 219 with 219 221

Figure 9-1 shows an example of how this is done with a sample IP datagram. IP datagrams are passed down to the SLIP software at layer 2 (a simplified one with only five bytes is shown here). There, they are framed by surrounding them with *END* characters (hexadecimal value C0h, shown with diagonal hatching). Special characters with hexadecimal values DBh and C0h are replaced by two-byte sequences. Note that the presence of the bracketing *END* characters forces the receiving device to see the noise byte (03h, in black) as a separate IP datagram, rather than part of either of the real ones. It will be rejected when passed up to the IP layer.

### **Problems and Limitations of SLIP**

SLIP’s simplicity does not come without costs. SLIP simply doesn’t provide many of the features and capabilities you really need on modern serial links. SLIP is most deficient in the following areas:

**Standardized Datagram Size Specification** SLIP’s maximum supported datagram size is not standardized and depends on each implementation. The usual default is 1,006 bytes, which becomes the maximum transmission unit (MTU) for the link (see Chapter 27). If a different size is used, you must program this into the IP layer.



**Figure 9-1: Operation of the Serial Line Internet Protocol (SLIP)** SLIP's only function is to frame data from layer 3 (usually IP datagrams) by surrounding them with END characters and replacing special characters as needed.

**Error Detection and Correction Mechanism** SLIP doesn't provide any way of detecting or correcting errors in transmissions. While such protection is provided at higher layers through IP header checksums and other mechanisms, it is a job traditionally also done at layer 2. The reason is that relying on those higher layers means that errors are detected only after an entire datagram has been sent and passed back up the stack at the recipient. Error correction can come only in the form of resending any datagrams that were corrupted. This is inefficient, especially because serial links are generally much slower than normal LAN links.

**Control Messaging** SLIP offers no way for the two devices to communicate control information that may be required to manage the link.

**Type Identification** Since SLIP includes no headers of its own, it is not possible to identify that SLIP is being used. While developed for IP, there is no reason why other layer 3 protocols could not be sent using SLIP (if you were running more than one internetworking protocol at the higher layers). However, without type identification, there is no way to mix datagrams from two or more layer 3 protocols on the same link.

**Address Discovery Method** Addressing isn't needed at layer 2 because there are only two devices in a point-to-point connection, so each device is obviously only sending to the other one. However, devices do need some way of learning each other's IP addresses for routing at layer 3. SLIP provides no method for this.

**Support for Compression** Compression would improve performance over serial lines that are otherwise slow compared to other technologies. SLIP provides no compression features. (Note, however, that modems usually do support compression at layer 1 for serial connections that use them.) A variant on SLIP called *Compressed SLIP* or *CSLIP* was created in the late 1980s, but it was not as widely deployed as regular SLIP.

**Security Features** SLIP lacks even basic security features, with no means for authenticating connections or encrypting data.

SLIP's many shortcomings have led most implementations to move from SLIP to the PPP, which is a much richer data link protocol for direct connections that resolves SLIP's problems. SLIP is now outdated. Still, SLIP continues to be used in many places. Simplicity is attractive, and people are famous for their inertia: If something is implemented and is working well, many will refuse to change unless they are forced to do so.

## Point-to-Point Protocol (PPP) Overview and Fundamentals

Even as SLIP was being documented, work was underway on a newer protocol that would provide full-featured IP transmission over direct links between pairs of devices. The result is *PPP*, which defines a complete method for robust data link connectivity between devices using serial lines or other physical layers. It includes numerous capabilities and features, including error detection, compression, authentication, and encryption.

The proliferation of serial links, especially for dial-up Internet access, has led to the widespread use of PPP. PPP is now one of the most popular layer 2 WAN technologies in the networking world, and has replaced SLIP as the standard for serial connections on all but legacy implementations. While most often associated with dial-up modem use, PPP can run across any similar type of physical layer link. For example, it is often used to provide layer 2 functionality on Integrated Services Digital Network (ISDN).

**NOTE** *Although PPP is called a protocol and is usually considered part of TCP/IP, it is really more a protocol suite, since its operation is based on procedures defined in many individual protocols. Alternatively, its components can be viewed as subprotocols within PPP, even though they are not usually called that in the standards.*

### Development and Standardization

Unlike SLIP, PPP was developed to be a complete protocol suite that would enable fully functional layer 2 connectivity to support not just IP, but the transmission of other network layer protocols as well.

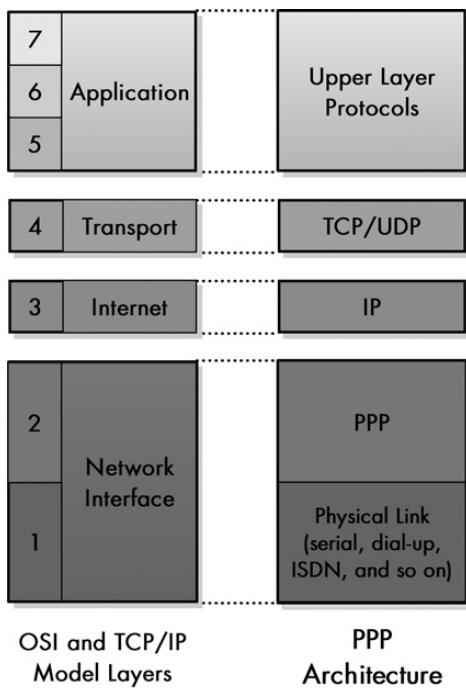
PPP's history goes back to the late 1980s, when SLIP was the de facto standard for serial IP implementations. The first formal IETF document related to PPP was RFC 1134 (1989). This RFC was not the standard itself, but a proposal for what would eventually be defined as the first main PPP standard, RFC 1171 (1990). RFC 1171 was revised several times, and several other documents were added to it to define the various protocols that compose the entire PPP suite.

**NOTE** *Rather than try to develop PPP from scratch, the IETF decided to base it on the ISO High-Level Data Link Control (HDLC) protocol, which was initially developed by IBM. HDLC is a derivative of the Synchronous Data Link Control (SDLC) Protocol. PPP's developers adapted its framing structure and some of its general operation from HDLC.*

## Function and Architecture

PPP is a connection-oriented protocol that enables layer 2 links over a variety of different physical layer connections. It is supported on both synchronous and asynchronous lines and can operate in half-duplex or full-duplex mode. It was designed to carry IP traffic, but is general enough to allow any type of network layer datagram to be sent over a PPP connection. As its name implies, PPP is designed for point-to-point connections between two devices, and it assumes that frames are sent and received in the same order.

PPP fits into TCP/IP in the network interface layer (link layer), as shown in Figure 9-2. PPP's operation follows a specific sequence, including a multistep Link Establishment phase that may include optional authentication.



**Figure 9-2: PPP location in the TCP/IP architecture** PPP is the interface between the IP and a physical link such as a serial line or dial-up networking connection. This corresponds to layer 2 in the OSI Reference Model.

## Advantages and Benefits

A list of PPP's strengths reads very much like a list of SLIP's weaknesses, as explained earlier in this chapter. Some of PPP's specific benefits include the following:

- A more comprehensive framing mechanism compared to the single *END* character in SLIP
- Specification of the encapsulated protocol to allow multiple layer 3 protocols to be multiplexed on a single link
- Error detection for each transmitted frame through the use of a cyclic redundancy check (CRC) code in each frame header
- A robust mechanism for negotiating link parameters, including the maximum frame size permitted

- A method for testing links before datagram transmission takes place and for monitoring link quality
- Support for authentication of the connection using multiple authentication protocols
- Support for additional optional features, including compression, encryption, and link aggregation (allowing two devices to use multiple physical links as if they were a single, higher-performance link)

The proliferation of serial links, especially for dial-up Internet access, has led to widespread use of PPP. It is now one of the most popular layer 2 WAN technologies in the networking world, and it has replaced SLIP as the standard for serial connections on all but legacy implementations. While most often associated with dial-up modem use, PPP can run across any similar type of physical layer link. For example, it is often used to provide layer 2 functionality on ISDN.

**KEY CONCEPT** PPP is a complete link layer protocol suite for devices using TCP/IP. It provides framing, encapsulation, authentication, quality monitoring, and other features that enable robust operation of TCP/IP over a variety of physical layer connections.

One key advantage of PPP is that it is *extensible*. Over the years, new protocols have been added to the suite in order to provide additional features or capabilities. For example, PPP is designed not to use just a single authentication protocol, but to allow a choice.

PPP's success has even led to the development of derivative protocols like PPP over Ethernet (PPPoE) and PPP over ATM (PPPoA). These derivatives actually layer PPP over existing data link layer technologies, which demonstrates how valued PPP's features are. Even when a layer 2 technology is already in use, you can apply PPP on top to provide authentication and management benefits for services like Digital Subscriber Line (DSL).

### **PPP Main Components**

At the highest level, PPP's functions can be broken down into several components. Each encompasses a general class of PPP functionality and is represented by either one protocol in the suite or a set of protocols. The PPP standard describes three main components of PPP:

**PPP Encapsulation Method** The primary job of PPP is to take higher-layer messages, such as IP datagrams, and encapsulate them for transmission over the underlying physical layer link. To this end, PPP defines a special frame format for encapsulating data for transmission, based on the framing used in HDLC. The PPP frame was designed to be small and contain only simple fields in order to maximize bandwidth efficiency and speed in processing.

**Link Control Protocol (LCP)** LCP is responsible for setting up, maintaining, and terminating the link between devices. It is a flexible, extensible protocol that allows many configuration parameters to be exchanged to ensure that both devices agree on how the link will be used.

**Network Control Protocols (NCPs)** PPP supports the encapsulation of many different layer 3 datagram types. Some of these require additional setup before the link can be activated. Once the general link setup is completed with LCP, control is passed to the NCP that is specific to the layer 3 protocol being carried on the PPP link. For example, when IP is carried over PPP, the NCP used is the PPP Internet Protocol Control Protocol (IPCP). Other NCPs are defined for supporting the Internetworking Packet Exchange (IPX) protocol, the NetBIOS Frames (NBF) protocol, and so forth.

The PPP encapsulation method and LCP are defined in the main PPP standard and some support standards; the NCPs are described in separate standard documents, one per NCP.

### **PPP Functional Groups**

While PPP's main components constitute much of the total package, I would add two additional functional groups. These represent some of the many extra protocols that have been added to the suite over time to support or enhance its basic operation:

**LCP Support Protocols** Several protocols in the PPP suite are used during the link negotiation process, either to manage it or to configure options. Examples include the authentication protocols Challenge Handshake Authentication Protocol (CHAP) and Password Authentication Protocol (PAP), which are used by LCP during the optional Authentication phase. These are discussed in Chapter 10.

**LCP Optional Feature Protocols** A number of protocols have been added to the basic PPP suite over the years to enhance its operation once a link has been set up and datagrams are being passed between devices. For example, the PPP Compression Control Protocol (CCP) allows compression of PPP data; the PPP Encryption Control Protocol (ECP) enables datagrams to be encrypted for security; and the PPP Multilink Protocol (PPP MP) allows a single PPP link to be operated over multiple physical links. These protocols often also require additional setup during link negotiation, so many of them define extensions (such as extra configuration options) that are negotiated as part of LCP.

**NOTE** *Each optional protocol is defined by a specific standards document, as you will see later in this chapter.*

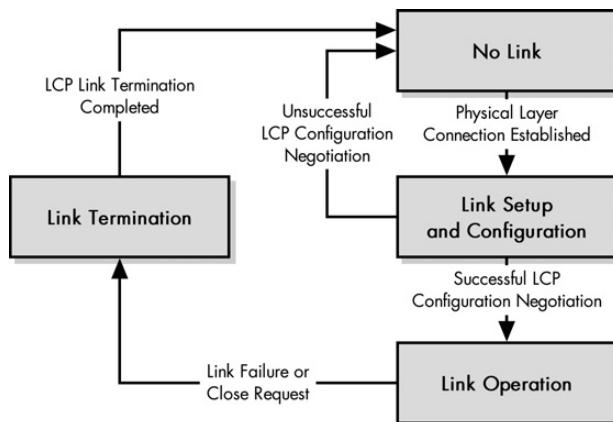
### **General Operation**

Although the PPP suite includes dozens of protocols, its general operation is really quite straightforward. Essentially, PPP involves the following three basic steps (see Figure 9-3):

1. **Link Setup and Configuration** Before the two devices can exchange information, they must make contact and set up a link between them. During link setup, the devices agree on all the parameters needed to manage the operation of the link. LCP begins this process and invokes the help of support protocols

as needed, for options like authentication. Once the link is set up, in order to complete link setup, the appropriate NCP is called for whatever layer 3 technology is being carried on the link.

2. **Link Operation** The devices use the link to send datagrams. Each device transmits by encapsulating layer 3 datagrams and sending them down to layer 1 to be transmitted. Each device receives by taking PPP frames sent up from its own physical layer, stripping off the PPP header and passing the datagram up to layer 3. Where appropriate, optional protocols are used at this stage to offer features such as compression (CCP).
3. **Link Termination** When either device decides that it no longer wants to communicate, it terminates the link.



**Figure 9-3: Overview of PPP operation** In simplest terms, PPP consists of only three basic steps: link setup, link operation, and link termination.

Link setup is by far the most complicated of these general steps, because it involves several substeps used to negotiate link parameters and options.

### **PPP Link Setup and Phases**

Before data can be exchanged on a PPP connection, a link must be set up between the two devices. As part of this setup task, a configuration process is undertaken whereby the devices configure the link and agree on the parameters for how data should be passed between them. Only when this is completed can frames actually pass over the link.

LCP is generally in charge of setting up and maintaining PPP links. LCP may invoke an authentication protocol (PAP or CHAP) when PPP is configured to use authentication. Once an LCP link has been opened, PPP invokes one or more NCPs for the layer 3 protocol being carried on the link. These perform any network-layer-specific configuration needed before the link can carry that particular network layer protocol.

The operation of a PPP link can be described as having a life of sorts: A PPP link is established, configured, used, and eventually terminated. The process of setting up, using, and closing a PPP link is described in the PPP standard as a series

of *phases* or *states*. This is a type of *finite state machine (FSM)*, which is a tool used to explain the operation of protocols. The general concept behind an FSM is described in the section discussing the finite state machine of the Transmission Control Protocol (TCP), in Chapter 47.

To better understand how PPP works, let's look at these phases and how the transition is made from one to the next during the lifetime of the link. For the sake of clarity, this description is based on an example for which Device A is a PC connecting via dial-up networking to Remote Host B (see Figure 9-4).

**NOTE** When we talk about a PPP link overall, we are talking about the status of the LCP connection between the two devices. Once an LCP link has been opened, each of the NCPs used on the link can be opened or closed independently of the overall PPP (LCP) link. You'll see how this works momentarily.

### Link Dead Phase

By design, the PPP link always begins and ends in the *Link Dead* phase. This phase represents the situation in which there is no physical layer link established between the two devices. The link remains here until the physical layer link is set up, at which point it proceeds to the *Link Establishment* phase.

In this example, when Device A is first turned on, there is no physical layer connection (modem connection) between it and Device B. Once the connection is made, the link can proceed to phase 2.

**NOTE** In a direct connection, such as a serial cable linking two PCs, the link may stay in the *Link Dead* phase for only a fraction of a second, until the physical layer connection is detected.

### Link Establishment Phase

The physical layer is now connected and LCP performs the basic setup of the link. Device A sends an LCP configuration request message to Device B over the physical link, specifying the parameters it wishes to use. If Device B agrees, it replies with an acknowledgment. If Device B doesn't agree, it returns a negative acknowledgment or rejection, telling Device A what it won't accept. Device A can then try a different configuration request with new parameters that Device B may accept. (This process is described in more detail in Chapter 10.)

If Device A and Device B eventually come to agreement, the link status is considered *LCP open* and will proceed to the *Authentication* phase. If they cannot agree, the physical link is terminated, and it returns to the *Link Dead* phase.

### Authentication Phase

In many cases, a device may require authentication before it will permit another device to connect. (This is usually the case when PPP is used for dial-up.) Authentication is not mandatory in PPP, however. When it is used, the appropriate authentication protocol (CHAP or PAP) is employed.

After successful authentication, the link proceeds to the *Network Layer Protocol* phase. If authentication is not successful, the link fails and transitions to the *Link Termination* phase.

## **Network Layer Protocol Phase**

Once the basic link has been configured and authentication has completed, the general setup of the LCP link is complete. Now, the specific configuration of the appropriate network layer protocol is performed by invoking the appropriate NCP, such as IPCP, IPXCP, and so forth.

Each particular network layer protocol whose NCP is successfully configured is considered to be open on the LCP link. More than one NCP can be open on a particular PPP link, and each can be closed independently when it is no longer needed. Once all necessary NCPs have been invoked, the link proceeds to the *Link Open* state, even if none of the NCPs were successfully opened.

**NOTE** Some PPP features require the negotiation of additional options between the two devices, which may perform their own link establishment process during the Network Layer Protocol phase. The PPP Compression Control Protocol (CCP) sets up data compression in this manner.

## **Link Open Phase**

In the Link Open state, the LCP link and one or more NCP links are open and operational. Data can be passed for each NCP that has been successfully set up.

The link can be terminated at any time by either device for a variety of reasons. These may include a user request (you click Disconnect when you want to log off your dial-up session); link quality problems (the modem hangs up on you due to line noise); or some other cause (you spend too much time in the bathroom and your ISP's idle timer logs you out). When any of these occur, the LCP link is broken, and the link transitions to the *Link Termination* phase.

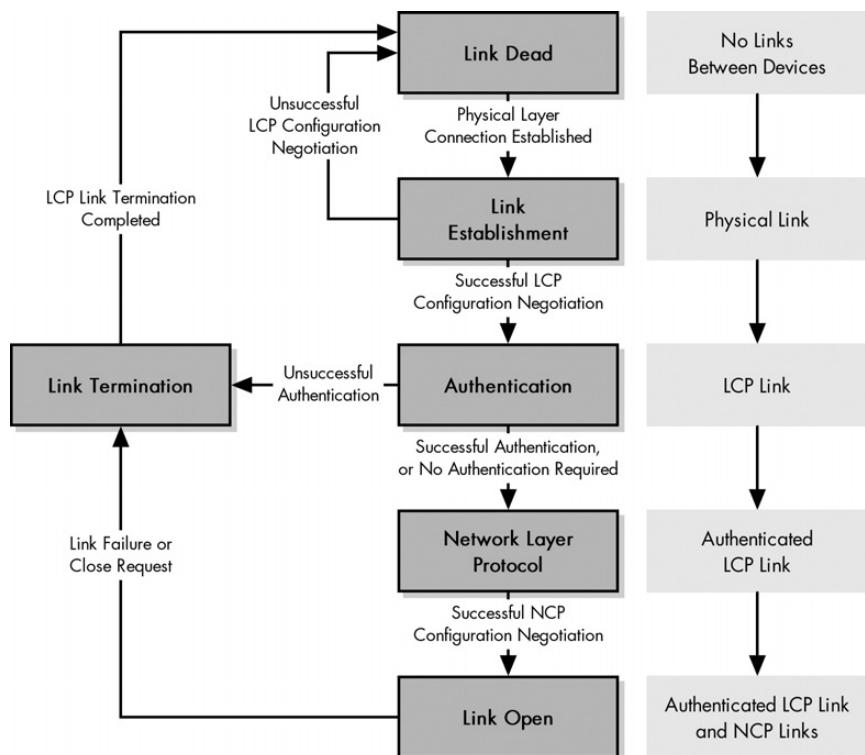
## **Link Termination Phase**

The device terminating the link sends a special LCP termination frame, and the other device acknowledges it. The link then returns to the Link Dead phase. If the termination was by request and the physical layer connection is still active, the PPP implementation should specifically signal the physical layer to terminate the layer 1 connection.

You should remember that the basic link is established by LCP, and NCP links are set up within the LCP link. Closing an NCP link does not cause the LCP link to be closed. Even if all NCPs are closed, the LCP link remains open. (Of course, no data can be passed until an appropriate NCP link is reestablished; a device is required to discard frames received that contain any layer 3 protocol that does not have an open NCP.) To terminate a PPP connection, only the LCP link needs to be terminated in the *Link Termination* phase; the NCPs do not need to be explicitly closed.

Figure 9-4 shows the PPP phases and the circumstances under which transitions occur between them. The PPP connection between two devices begins in the Link Dead state and proceeds through three intermediate phases until the link is fully

opened. It remains in the stable Link Open phase until terminated. The lighter boxes show the corresponding change in the status of the PPP link as transitions are made between phases.



**Figure 9-4: PPP phases** A PPP connection follows a mainly linear sequence of transitions from the Link Dead Phase through the Link Open Phase.

Table 9-1 summarizes the PPP phases; the LCP Link Status and NCP Link Status columns show the status of the link as the phase starts.

### PPP Standards

While it makes sense for different parts of PPP to be covered in different standards, this does make it much harder to learn how PPP works. Also, literally *dozens* of RFCs cover PPP's main operation, its various protocols, and other related issues. You can find most of them by consulting a master list of RFCs and searching for the string "PPP," but you will find them in numerical (RFC number) order, which isn't very meaningful in terms of how the protocols are used. You also have to differentiate between the ones that are current and those that are obsolete.

**Table 9-1: PPP Phases**

Phase/State	Phase Summary	LCP Link Status Upon Entry to Phase	NCP Link Status Upon Entry to Phase	Transition Requirement	Transition to Phase
Link Dead	Default state; physical layer not connected.	Closed	Closed	Successful physical layer connection	Link establishment
Link Establishment	Physical layer connected, basic configuration of link performed by LCP.	Closed	Closed	Successful negotiation	Authentication
				Unsuccessful negotiation	Link dead
Authentication	Basic link is now opened, and optional authentication of device is performed.	Open	Closed	Successful authentication or no authentication required	Network layer protocol
				Unsuccessful authentication	Link termination
Network Layer Protocol	One or more NCPs open an NCP link within the LCP link.	Open	Closed	All NCPs opened	Link open
Link Open	Link is open and operating normally.	Open	Open	Link failure or close request	Link termination
Link Termination	LCP link is shut down.	Open	Open		Link dead

Table 9-2 lists the most important and interesting PPP-related RFCs. To make it easier to see what the RFCs are about, I have organized them into five groups, as follows:

**Core** These are PPP's main documents. They cover the basic operation of PPP including the PPP LCP and encapsulation of datagrams.

**LCP Support** These protocols support the basic operation of LCP. I've only included the ones that provide authentication services during link startup.

**NCPs** These protocols negotiate parameters specific to various layer 3 protocols carried over PPP.

**Features** These protocols define optional features used with PPP, such as compression and encryption.

**Applications and Miscellaneous** These are the protocols that describe how PPP can be adapted to run over particular types of links or that don't really fit into any of the previous groups.

Within each group, the RFCs are listed in numerical order, which is also date order. Only the most recent RFC is listed, not earlier ones that were made obsolete (with the exception of RFC 1334, which, despite being made obsolete, is still important).

**Table 9-2: PPP Standards**

<b>Group</b>	<b>RFC Number</b>	<b>Standard Name</b>	<b>Description</b>
Core	1570	PPP LCP Extensions	Defines two features for LCP that allow devices to identify each other and for each device to tell the other how much time remains in the current session.
	1661	The Point-to-Point Protocol (PPP)	Base standard for PPP. Describes PPP architecture, general operation (including the process of link establishment, maintenance, and termination), and details of LCP.
	1662	PPP in HDLC-like Framing	Defines the specific framing method for PPP frames, based on that used in HDLC. This standard can be considered a companion to the main PPP standard, RFC 1661.
LCP Support	1334	PPP Authentication Protocols	Defines the two PPP authentication protocols: PAP and CHAP. Note that RFC 1994 obsoletes RFC 1334, but does not discuss the PAP. (That tells you that the IETF doesn't think highly of PAP; see Chapter 10 for more on this.)
	1994	PPP Challenge Handshake Authentication Protocol (CHAP)	Updates the information about CHAP provided in RFC 1334.
NCPs	1332	The PPP Internet Protocol Control Protocol (IPCP)	The NCP for IP.
	1377	The PPP OSI Network Layer Control Protocol (OSINLCP)	The NCP for OSI protocol suite network layer protocols, such as CNLP, ES-IS, and IS-IS.
	1378	The PPP AppleTalk Control Protocol (ATCP)	The NCP for the AppleTalk protocol.
	1552	The PPP Internetworking Packet Exchange Control Protocol (IPXCP)	The NCP for the Novell IPX protocol.
	2043	The PPP SNA Control Protocol (SNACP)	The NCP for IBM's Systems Network Architecture (SNA).
	2097	The PPP NetBIOS Frames Control Protocol (NBFCP)	The NCP for NetBIOS Frames (NBF, also commonly called NetBEUI).
	2472	IP Version 6 over PPP	The NCP for IPv6: the IPv6 Control Protocol (IPv6CP).
Features	1962	The PPP Compression Control Protocol (CCP)	Defines a mechanism for compressing data sent over PPP links to improve performance. This standard describes how compression is negotiated between two devices on a PPP link. It is used in conjunction with several compression algorithms that actually do the compression of data.
	1968	The PPP Encryption Control Protocol (ECP)	Defines a mechanism for encrypting data sent over PPP links to improve performance. This standard describes how encryption is negotiated between two devices. It is used with several encryption algorithms.

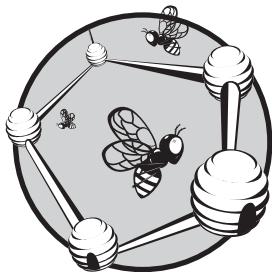
*(continued)*

**Table 9-2:** PPP Standards (continued)

<b>Group</b>	<b>RFC Number</b>	<b>Standard Name</b>	<b>Description</b>
Features, <i>continued</i>	1989	PPP Link Quality Monitoring	Defines a protocol that lets PPP devices generate reports to each other about the quality of the link.
	1990	The PPP Multilink Protocol (MP)	Defines a method for running PPP over a set of aggregated links, thereby allowing two devices to use multiple low-bandwidth links as a single, high-bandwidth virtual link.
	2125	The PPP Bandwidth Allocation Protocol (BAP)/The PPP Bandwidth Allocation Control Protocol (BACP)	Defines two support protocols that manage the allocation of bandwidth in links aggregated using PPP MP.
Applications and Miscellaneous	1618	PPP over ISDN	Describes application particulars for running PPP over ISDN links.
	1973	PPP in Frame Relay	Describes how PPP may be modified to run over Frame Relay at layer 2.
	2290	Mobile-IPv4 Configuration Option for PPP IPCP	Defines changes to the PPP Internet Protocol Control Protocol (IPCP) to support Mobile IP.
	2364	PPP over AAL5	Defines a method for sending PPP frames over AAL5 (ATM), commonly called PPPoA.
	2516	A Method for Transmitting PPP over Ethernet (PPPoE)	Defines a technique for encapsulating PPP frames over Ethernet (PPPoE).
	2615	PPP over SONET/SDH	Discusses how to encapsulate PPP frames over SONET/SDH links.

# 10

## **PPP CORE PROTOCOLS: LINK CONTROL, NETWORK CONTROL, AND AUTHENTICATION**

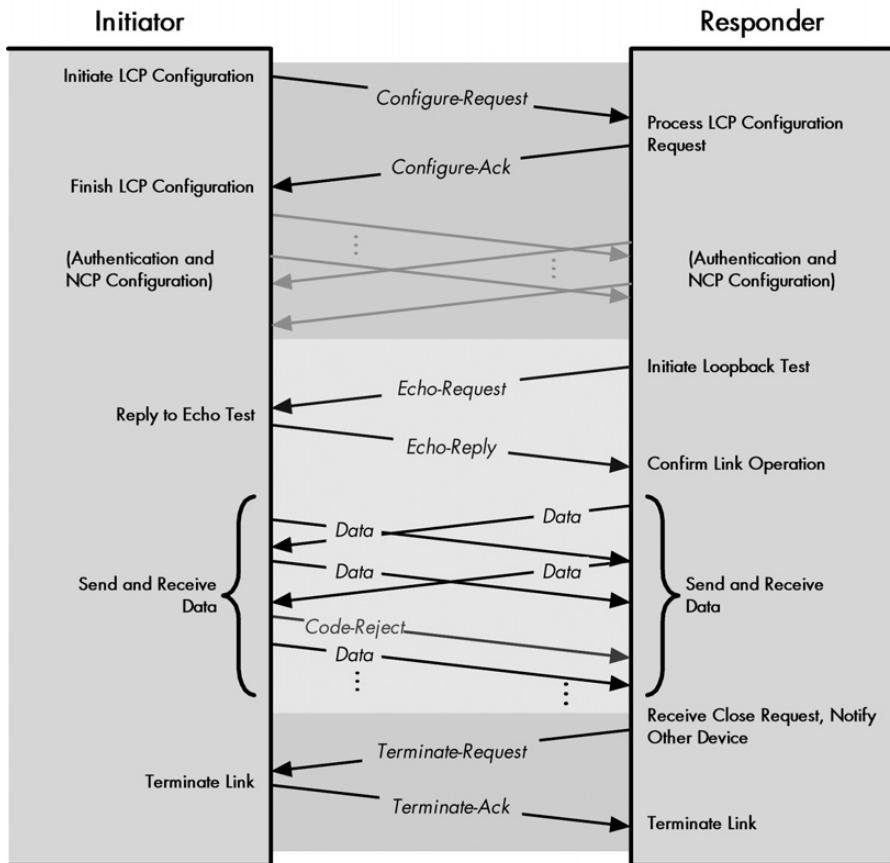


This chapter describes the protocols responsible for PPP link setup and basic operation, including Link Control Protocol (LCP) and the Network Control Protocols (NCPs) used to configure PPP for different layer 3 protocols. I also discuss the two PPP authentication protocols, Password Authentication Protocol (PAP) and Challenge Handshake Authentication Protocol (CHAP), which are used to provide authentication during link setup.

### **Link Control Protocol (LCP)**

Of all the PPP suite protocols, LCP is the most important. It is responsible for PPP's overall successful operation, and plays a key role in each PPP link stage: configuration, maintenance, and termination (as discussed in Chapter 9). Link configuration is performed during the initial link establishment phase; link maintenance occurs while the link is open, and link termination happens in the link termination phase.

Figure 10-1 provides an overview of many of the message exchanges performed by LCP during different phases of a PPP connection. Link configuration is shown here as a simple exchange of a Configure-Request and Configure-Ack. After subsequent exchanges using other PPP protocols to authenticate and configure one or more NCPs, the link enters the link open phase. In this example, Echo-Request and Echo-Reply messages are first used to test the link, followed by the sending and receiving of data by both devices. One Data message is shown being rejected due to an invalid Code field. Finally, the link is terminated using Terminate-Request and Terminate-Ack messages.



**Figure 10-1: PPP Link Control Protocol (LCP) message exchanges** This diagram shows the different message exchanges performed by LCP during link configuration, maintenance, and termination.

## LCP Packets

Devices use LCP to control the PPP link by sending LCP messages across the physical link between them. These messages are called both *LCP packets* and *LCP frames*. Although the standard uses *packet*, the term *frame* is preferred because layer 2 messages are normally called frames. The main PPP document defines 11

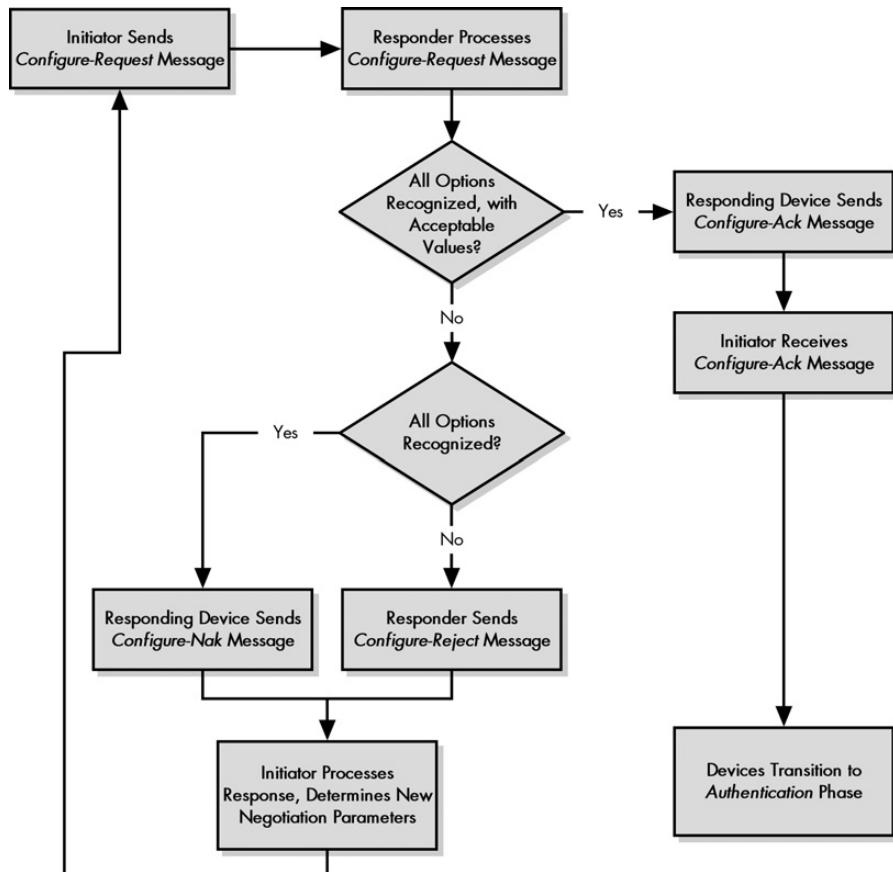
different LCP frames, which are divided into three groups that correspond to the three link stages. Four LCP frame types are used for link configuration, five for maintenance, and two for termination.

In the following section I'll discuss each of the three major functions of LCP and how the frames are used in each. (Chapter 12 describes the frame formats for the packets themselves.)

## LCP Link Configuration

Link configuration is arguably LCP's most important job in PPP. During the link establishment phase, the two physically connected devices exchange LCP frames that help them negotiate the conditions under which the link will operate. Figure 10-2 shows the entire procedure.

The process begins with the initiating device (Device A) creating a Configure-Request frame that contains a variable number of configuration options that it wants to see set up on the link. This is basically Device A's "wish list" for how it wants the link created.



**Figure 10-2: PPP LCP link configuration process** The negotiation process undertaken to configure the link by LCP. This process begins when the PPP link enters the link establishment phase. After successful configuration, the connection transitions to the authentication phase.

RFC 1661, the main PPP document, defines a number of different configuration options that the initiator can specify in this request. Any one of these can be included and if so, filled in with the value corresponding to what Device A wants for that option. If absent, Device A isn't requesting that option. The six options are as follows:

**Maximum Receive Unit (MRU)** Lets Device A specify the maximum size datagram it wants the link to be able to carry.

**Authentication Protocol** Device A can indicate the type of authentication protocol it wishes to use (if any).

**Quality Protocol** If Device A wants to enable quality monitoring on the link, what quality monitoring protocol to use (though there is only one currently defined: LQR).

**Magic Number** Used to detect looped-back links or other anomalies in the connection.

**Protocol Field Compression** Allows Device A to specify that it wants to use “compressed” (8-bit) Protocol fields in PPP data frames instead of the normal 16-bit Protocol field. This provides a small (one byte) but free savings on each PPP frame. (Note that this has nothing to do with the compression feature offered by Compression Control Protocol, or CCP; see the PPP general frame format discussion in Chapter 12 for more on this feature.)

**Address and Control Field Compression (ACFC)** The same as Protocol Field Compression, but used to compress the Address and Control fields for small bandwidth savings. (See the PPP general frame format topic in Chapter 12 for more.)

Other options may also be added to this list by optional feature protocols. For example, Multilink PPP (Chapter 11) adds several options that must be negotiated during link setup.

The other device (Device B) receives the Configure-Request and processes it. It then has the following three choices of how to respond:

- If every option in it is acceptable, Device B sends back a Configure-Ack (acknowledge) frame. The negotiation is complete.
- If Device B recognizes all the options that Device A sent as valid and is capable of negotiating, but it doesn't accept the values, Device B returns a Configure-Nak (negative acknowledge) frame. This message includes a copy of each configuration option that Device B found unacceptable.
- If any of the options that Device A sent were either unrecognized by Device B or represent ways of using the link that Device B considers not only unacceptable, but not even subject to negotiation, it returns a Configure-Reject containing each of the objectionable options.

The difference between a Configure-Nak and a Configure-Reject is that the former is like Device B saying, “I don't accept your terms, but I'll discuss,” while the latter is Device B basically saying, “No way Jose!” For example, if Device A tries to

request PAP as the authentication protocol, but Device B wants to use CHAP, it will send a Configure-Nak. If Device B doesn't support authentication at all, it will send a Configure-Reject.

**NOTE** Even after receiving a rejection, Device A can retry the negotiation with a new Configure-Request.

### **LCP Link Maintenance**

Once the link has been negotiated, LCP passes control to the appropriate authentication and NCP protocols (as discussed below). Eventually the link setup will complete and go into the open state, at which point, LCP messages can then be used by either device to manage or debug the link, as follows:

**Code-Reject and Protocol-Reject** These frame types are used to provide feedback when one device receives an invalid frame due to either an unrecognized LCP code (LCP frame type) or a bad protocol identifier.

**Echo-Request, Echo-Reply, and Discard-Request** These frames can be used for testing the link.

### **LCP Link Termination**

When the link is ready to be shut down, LCP terminates it. The device initiating the shutdown (which may not be the one that initiated the link in the first place) sends a Terminate-Request message. The other device replies with a Terminate-Ack message. A termination request indicates that the device sending it needs to close the link. This is a request that cannot be denied.

### **Other LCP Messages**

The standard RFC 1570, “PPP LCP Extensions,” also defines two new LCP message types. The Identification message is used to allow a device to identify itself to its peer on the link. The Time-Remaining message lets one device tell the other how much time remains in the current session.

Many of the other protocols used in PPP are modeled after LCP. They use the same basic techniques for establishing protocol connections, and send and receive a subset of LCP message types. They also exchange configuration options in a similar manner.

## **The Network Control Protocols (IPCP, IPXCP, NBFCP, and Others)**

Although PPP was originally created to carry IP datagrams, its designers realized that it could easily carry data from many types of network layer protocols, and that, on some networks, it might even be advantageous to let it carry datagrams from different layer 3 protocols simultaneously.

Allowing PPP to support multiple network layer protocols would require it to have knowledge of each one's idiosyncrasies. If you used only LCP for link configuration, the device would need to know all the unique requirements of each layer 3 protocol. This would also require you to update LCP constantly as new layer 3 protocols were defined and as new parameters were defined for existing ones.

To eliminate this potential issue, PPP takes a modular approach to link establishment. LCP performs the basic link setup, and after (optional) authentication, invokes an *NCP* that is specific to each layer 3 protocol that is to be carried over the link. The NCP negotiates any parameters that are unique to the particular network layer protocol, and more than one NCP can be run for each LCP link (see the discussion of PPP link setup and phases in Chapter 9).

Each of the common network layer technologies has a PPP NCP defined for it in a separate RFC. The most common ones, “The PPP Internet Protocol Control Protocol (IPCP),” “The PPP Internetworking Packet Exchange Control Protocol (IPXCP),” and “The PPP NetBIOS Frames Control Protocol (NBFCP),” are NCPs for IP, IPX, and NBF (also called NetBEUI), respectively. A separate NCP is also defined for IP version 6, the “PPP IP Version 6 Control Protocol (IPv6CP).”

## ***Operation of NCPs***

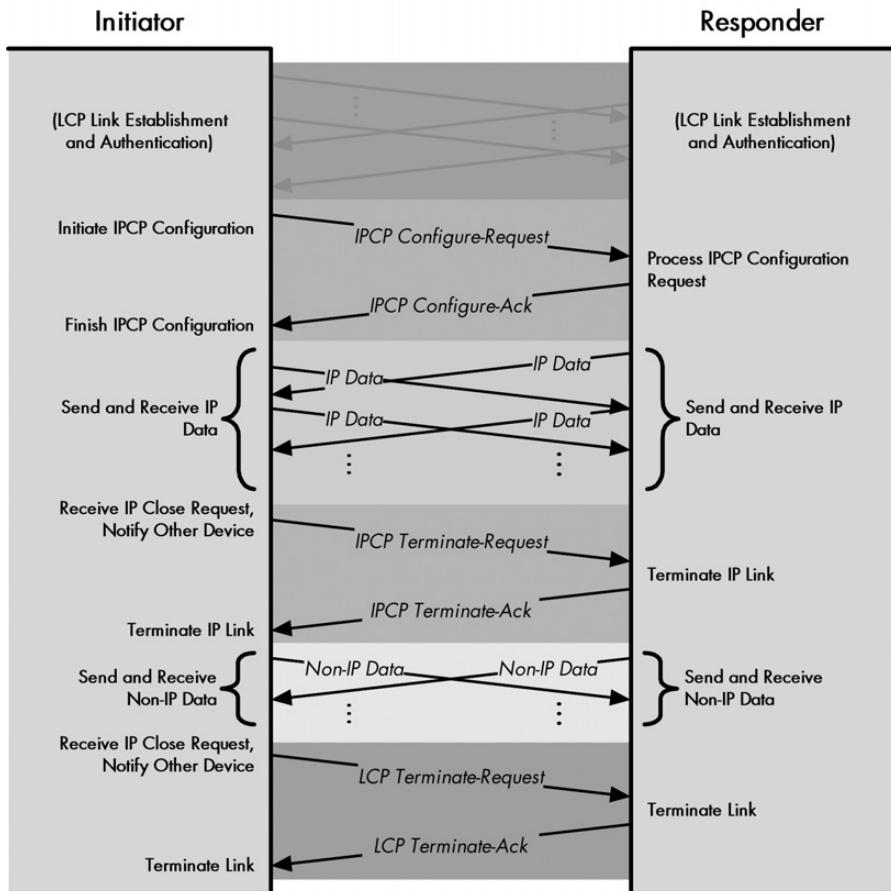
Each NCP operates very much like a light version of LCP, as you can see in Figure 10-3. (To see the similarities, you should compare Figure 10-3 to Figure 10-1, which shows the messaging for LCP.) Like LCP, each NCP performs functions for link setup, maintenance, and termination, except that it deals only with its particular type of NCP link and not the overall LCP link. Each NCP uses a subset of the following seven of the message types defined in LCP in very much the same way that LCP uses each message type of the same name, as shown for each of these three main link activities:

**Link Configuration** The process of setting up and negotiating the parameters of a particular NCP link (once an LCP link is established) is accomplished using Configure-Request, Configure-Ack, Configure-Nak, and Configure-Reject messages as discussed for LCP (except that these are particular to each NCP). The configuration options are the network layer protocol parameters being negotiated.

**Link Maintenance** Code-Reject messages can be sent to indicate invalid code values (NCP frame types).

**Link Termination** An NCP link can be terminated using Terminate-Request and Terminate-Ack messages. But remember that NCP links are set up within an LCP link, and that there can be more than one NCP link open. Closing NCP links doesn't terminate the LCP link. (NCP links do not need to be closed when an LCP link is terminated.)

Figure 10-3 shows how the overall operation of the NCPs, such as IPCP, is very similar to that of LCP. Once LCP configuration (including authentication) is complete, IPCP Configure-Request and Configure-Ack messages are used to establish an IPCP link. IP data can then be sent over the link. If the IPCP connection is no longer needed, it may be terminated, after which the LCP link remains open for other types of data to be transmitted. It is not necessary, however, to explicitly terminate the IPCP link before terminating the LCP connection.



**Figure 10-3: PPP IP Control Protocol (IPCP) message exchanges** The message exchanges, performed to configure and terminate IPCP, are quite similar to those used for LCP.

**KEY CONCEPT** Once the primary PPP link is established using LCP, each network layer protocol to be carried over the link requires the establishment of the appropriate NCP link. The most important of these is the *PPP Internet Protocol Control Protocol (IPCP)*, which allows IP datagrams to be carried over PPP.

## **The Internet Protocol Control Protocol (IPCP): An Example NCP**

Let's look at the NCP for IP: IPCP. When PPP is set up to carry IP datagrams, IPCP is invoked in the network layer protocol phase to set up an IP NCP link between the two devices. The setup is carried out using the four Configure- messages. For IP, two configuration options can be specified in an IPCP Configure-Request message:

**IP Compression Protocol** Allows devices to negotiate the use of Van Jacobson TCP/IP header compression, which shrinks the size of TCP and IP headers to save bandwidth. This is similar in concept to the Protocol-Field-Compression and ACFC options in LCP.

**IP Address** Allows the device sending the Configure-Request message either to specify an IP address it wants to use for routing IP over the PPP link or to request that the other device supply it with one. This is most commonly used for dial-up networking links.

Once configuration is complete, data can be sent for the layer 3 protocol corresponding to the NCP negotiated. This is indicated by using the appropriate value for the Protocol field in PPP data frames containing that layer 3 data.

## **PPP Authentication Protocols: PAP and CHAP**

PPP was designed to provide layer 2 connectivity over a variety of serial links and other physical layer technologies, some of which introduce more security concerns than others. For example, suppose you connect two machines in your office with a serial cable and want to run PPP between them. When one of these initiates a PPP link with the other, you don't really need to worry about who's calling. On the other hand, consider an Internet service provider (ISP) using PPP for remote dial-in users. They generally want to allow only their customers to connect.

The PPP protocol suite allows for the use of an optional authentication protocol when devices negotiate the basic link setup. The PPP suite initially defined two such protocols: PAP and CHAP. Once an LCP link is set up between two devices, a series of authentication messages are sent using these protocols to verify the identity of the device initiating the link. Only if authentication is successful can the link configuration proceed.

### **PAP**

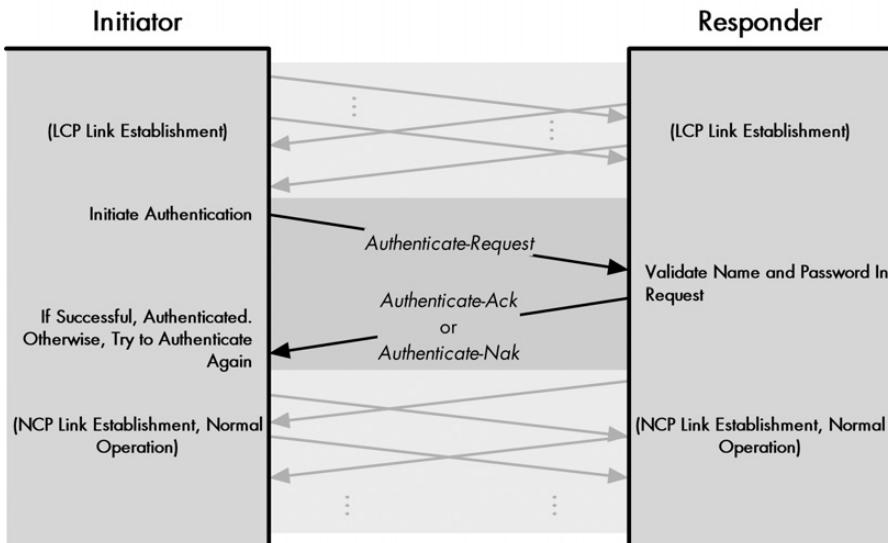
PAP is a very straightforward authentication scheme, consisting of only two basic steps, as shown in Figure 10-4.

**Authentication Request** The initiating device sends an Authenticate-Request message that contains a name and a password.

**Authentication Reply** The responding device looks at the name and password and decides whether to accept the initiating device and continue setting up the link. If so, it sends back an Authenticate-Ack message. Otherwise, it sends an Authenticate-Nak message.

PAP is another example of something that is just too simple for its own good. Chief among its flaws is that it transmits the user name and password in clear text across the link. This is a big “no-no” because eavesdroppers can get the password.

PAP also provides no protection against various security attacks. For example, an unauthorized user could try different passwords indefinitely until he discovered the correct one. PAP also puts control of the authentication squarely on the shoulders of the initiating device (usually a client machine), which is not considered desirable, because this is normally a server function that administrators prefer to manage.



**Figure 10-4: PAP authentication** PAP uses a simple exchange of a request containing name and password information and a reply indicating whether or not authentication was successful.

## CHAP

The most important difference between PAP and CHAP is that CHAP doesn’t transmit the password across the link. When using PAP, the initiator (calling client) sends the authenticator (generally the server that is deciding whether to grant authentication) a message saying essentially, “Here’s the password I know; see if it matches yours.” Each device uses the password to perform a cryptographic computation, and then checks to see if it gets the same result. If so, they know they have the same password.

In CHAP, a basic LCP link is first set up between the initiator and authenticator. The authenticator then takes charge of the authentication process, using a technique called a *three-way handshake*.

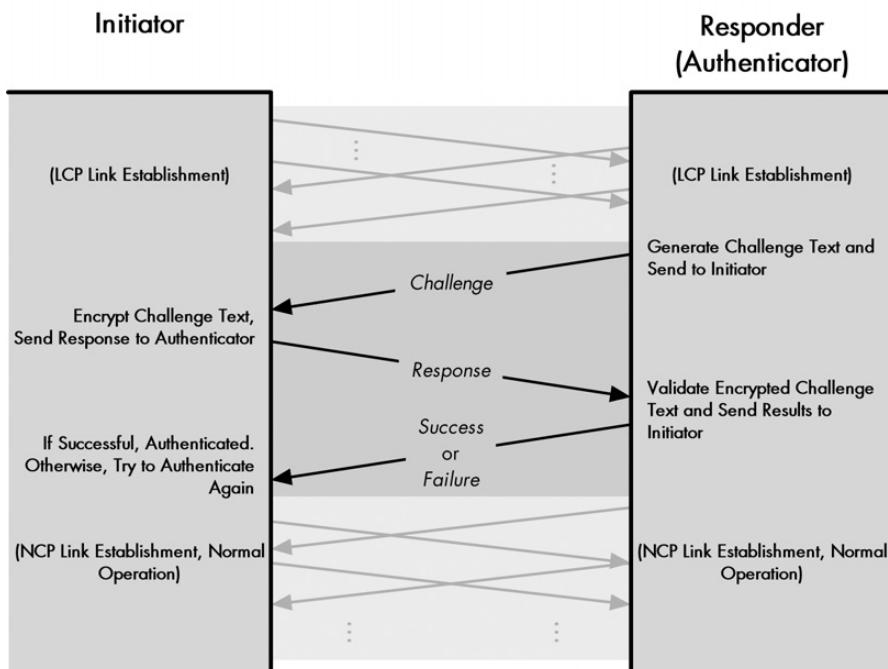
**NOTE** Three-way handshakes are a fairly common general authentication procedure. The same basic technique is used, for example, in shared key authentication on IEEE 802.11 wireless networking.

The three-way handshake steps are as follows (see Figure 10-5):

**Challenge** The authenticator generates a frame called a Challenge and sends it to the initiator. This frame contains a simple text message (sometimes called the *challenge text*). The message has no inherent special meaning, so it doesn't matter if anyone intercepts it. The important thing is that after receipt of the Challenge, both devices have the same Challenge message.

**Response** The initiator uses its password (or some other shared secret that the authenticators also know) to encrypt the challenge text. It then sends the encrypted challenge text as a Response back to the authenticator.

**Success or Failure** The authenticator performs the same encryption on the challenge text that the initiator did. If the authenticator gets the same result that the initiator sent it in the Response, it knows that the initiator had the right password when it did its encryption, so the authenticator returns a Success message. Otherwise, it sends a Failure message.



**Figure 10-5: PPP Challenge Handshake Authentication Protocol (CHAP) authentication** CHAP uses a three-way handshake beginning with a Challenge from the authenticating device. This message is encrypted and returned to the authenticating device, which checks to see if the device trying to authenticate used the correct password (or other shared secret).

The beauty of this is that it verifies that the two devices have the same shared secret, but it doesn't require them to send the secret over the link. The Response is calculated based on the password, but the content of the Response is encrypted, and thus it's much harder to derive the password from. CHAP also provides protection against replay attacks, whereby an unauthorized user captures a message and tries to send it again later on. This is done by changing an identifier in each message and varying the challenge text. Also, in CHAP, the server controls the authentication process, not the client that is initiating the link.

**KEY CONCEPT** PPP supports two authentication protocols: *PAP* and *CHAP*. PAP is a simple request-and-reply authentication protocol that is widely considered to be inadequate because it sends the user name and password in clear text and provides little protection against many security concerns. CHAP uses a three-way handshake procedure and is preferred over PAP in most implementations.

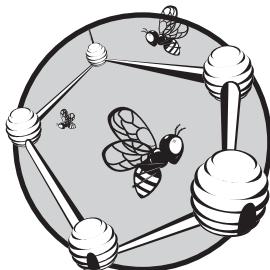
CHAP is not perfect, but it's much better than PAP. In fact, the IETF made a rather strong statement in this regard when it revised the original RFC that described PAP and CHAP to include only CHAP in the new standard. Despite this, PAP is still used in some applications because it is simple. PAP can suffice in situations where security is not a big deal, but CHAP is a much better choice.

**NOTE** *Incidentally, in addition to PAP and CHAP, it is possible to use proprietary authentication schemes. This requires that the appropriate configuration option values be programmed into LCP for placement in the Authentication Protocol configuration option.*



# 11

## PPP FEATURE PROTOCOLS



Point-to-Point Protocol (PPP) is the standard for data link layer connectivity over serial links because its core protocols provide a solid operational foundation, as you saw in Chapter 10. However, PPP's popularity is based not just on its highly capable link establishment and management features, but it also has a number of very useful features that provide important security and performance benefits to network users.

In this chapter, I describe the protocols that implement several of the most common extra features in PPP. I begin with a discussion of PPP link quality monitoring. I describe the sets of protocols used to configure and implement data compression and data encryption. I then discuss the PPP Multilink Protocol (MP, MLPPP), which allows PPP to bundle multiple low-speed links into a single high-speed link. I also cover the *Bandwidth Allocation Protocol* (*BAP*) and *Bandwidth Allocation Control Protocol* (*BACP*), which are used to manage the operation of MLPPP.

## **PPP Link Quality Monitoring and Reporting (LQM, LQR)**

PPP includes optional authentication in recognition of the varying *security* needs of the many different kinds of links over which PPP may operate. These links also differ greatly in terms of their *quality*. Just as you don't need to worry about authentication much when two machines are linked with a short cable, you also can feel pretty confident that data sent between them is going to arrive intact. Now contrast that with a PPP session established over a long-distance telephone call. For that matter, how about PPP over a dial-up call using an analog cellular phone?

PPP includes in its basic package a provision for detecting errors in sent frames, and higher-layer protocols like TCP also include methods of providing robustness on noisy lines. These techniques allow a link to tolerate problems, but provide little in the way of useful information about what the status of the link is. In some situations, devices may want to be able to keep track of how well the link is working, and perhaps take action on it. For example, a device experiencing too many errors on a dial-up connection might want to cut off and retry a new call. In some cases, a device might want to try an alternate method of attachment if the current physical link is not working well.

Recognizing this need, the PPP suite includes a feature that allows devices to analyze the quality of the link between them. This is called *PPP Link Quality Monitoring* or *LQM*. PPP is set up generically to allow any number of different monitoring functions to be used, but at present, there is only one, called *Link Quality Reporting (LQR)*. LQR allows a device to request that its peer (the other device on the link) keep track of statistics about the link and send periodic reports about them.

### **LQR Setup**

Before LQR can be used, it must be set up, which is done by LCP as part of the negotiation of basic link parameters in the Link Establishment phase (see Chapter 10). The device opening the link requests link monitoring by including the Quality Protocol configuration option in its Configure-Request frame. The configuration option also specifies a *reporting period* that indicates the longest period of time that the requesting device wants to go between receiving reports.

Assuming that the negotiation is successful, LQR will be enabled. A number of counters are set up that keep track of various link statistics, and a timer is used to regulate the sending of quality reports over the link. Each time the timer expires, a link quality report is generated and sent in a PPP frame over the link using the special PPP Protocol field hexadecimal value 0xC025.

Each counter holds information about a different statistic regarding the use of the link. Each counter is reset to zero when LQR is set up and then incremented each time a transmission is made or an event occurs that is relevant to the counter. The statistics tracked include the following:

- The number of frames sent or received
- The number of octets (bytes) in all frames sent or received
- The number of errors that have occurred

- The number of frames that had to be discarded
- The number of link quality reports generated

These counters are reset only at the start of the link, so they contain figures that are kept cumulatively over the life of the connection. The counters can be used in the absolute sense, meaning that the counter value itself is reported.

Alternatively, they can be expressed as relative (or *delta*) values, which represent the change since the last report. This is done when a report is received, simply by subtracting the previous report's numbers from the ones in the current report.

## **Using Link Quality Reports**

LQR specifies the quality reporting mechanism, but not specific standards for link quality, since these are so implementation-dependent. Based on the numbers in these reports, a device can decide for itself what conclusions to draw about link quality and what action to take, if any. Here are some possible behaviors:

- Some devices might decide to shut down a link if the absolute number of errors seen in any report reaches a certain threshold.
- Some might look at the trend in successive reporting periods and take action if they detect certain trends, such as an increase in the rate of discarded frames.
- Some devices might just log the information and take no action at all.

**NOTE** *LQR aggregates its statistics for all higher-layer protocols transmitted over a particular link. It doesn't keep track of statistics for different higher-layer protocols separately, which makes sense, since the quality of the link shouldn't vary from one higher-layer protocol to the next.*

## **PPP Compression Control Protocol (CCP) and Compression Algorithms**

PPP is primarily used to provide data link layer connectivity to physical serial links. One of the biggest problems with serial links compared to many other types of layer 1 connections is that they are relatively slow. Consider that while 10 Mbps regular Ethernet is considered sluggish by modern LAN standards, it is actually much faster than most serial lines used for WAN connectivity, which can be ten, one hundred, or even one thousand times slower.

One way to improve performance over serial links is to compress the data sent over the line. Depending on the data transferred, this can double the performance compared to uncompressed transmissions, and can, in some cases, do even better than that. For this reason, many hardware devices include the ability to compress the data stream at the physical layer. The best example of this is probably the set of compression protocols used on analog modems.

Some physical links don't provide any compression capabilities, but could still benefit from it. To this end, an optional compression feature was created for PPP. It is implemented using the following two distinct protocol components:

**PPP Compression Control Protocol (CCP)** This protocol is responsible for negotiating and managing the use of compression on a PPP link.

**PPP Compression Algorithms** A set of compression algorithms that perform the actual compression and decompression of data. Several of these are defined in RFCs. In addition, it is possible for two devices to negotiate the use of a proprietary compression method if they want to use one that isn't defined by a public standard.

**KEY CONCEPT** PPP includes an optional compression feature that can improve performance over slow physical links. A variety of different compression algorithms are supported. To enable compression, both devices on a PPP link use the *PPP Compression Control Protocol (CCP)* to negotiate a compression algorithm to use. The compression algorithm is then used to compress and decompress PPP data frames.

## CCP Operation: Compression Setup

When most people talk about compression in PPP, they mention CCP, which is considered “the” compression protocol for PPP. However, CCP is actually used only to configure and control the use of compression; in fact, the algorithms do the real work of compressing and decompressing. This separation of powers provides flexibility, because it allows each implementation to choose what type of compression it wants to use.

CCP is analogous to the Network Control Protocols (NCPs) that negotiate parameters specific to a network layer protocol sent on the link. An NCP lets two devices decide how they will carry layer 3 traffic, such as how Internet Protocol Control Protocol (IPCP) lets the devices determine how to carry IP. CCP lets two devices decide how they will compress data, in the same basic way.

Similarly, just as each NCP is like a “light” version of LCP, CCP is like a light version of LCP. It is used to set up a compression connection called a *CCP link* within an LCP link between two devices. Once established, compressed frames can be sent between the two devices. CCP also provides messaging capabilities for managing and eventually terminating a CCP link. Again, this is very similar to how each network layer protocol sets up an NCP link within LCP. A CCP link is maintained independently of any NCP links.

CCP uses the same subset of seven LCP message types that the NCPs use, and it adds two additional ones. The use of these messages for each of the life stages of a CCP link is as follows (this should look familiar if you've read about how the NCPs and LCP work in Chapter 10):

**Link Configuration** Like the NCPs, compression configuration is done once CCP reaches the *network layer protocol* phase. The process of setting up compression and negotiating parameters is accomplished using Configure-Request, Configure-Ack, Configure-Nak, and Configure-Reject messages, just as it is for LCP, except the configuration options are particular to CCP.

**Link Maintenance** Code-Reject messages can be sent to indicate invalid code values in CCP frames. The two new message types are Reset-Request and Reset-Ack, which are used to reset the compression (the CCP link) in the event of a detected failure in decompression.

**Link Termination** A CCP link can be terminated using Terminate-Request and Terminate-Ack. Again, remember that, like the NCP links, the CCP link is set up within an LCP link, and closing it doesn't terminate the LCP link, which controls PPP overall.

## ***CCP Configuration Options and Compression Algorithms***

CCP configuration options are used only to negotiate the type of compression to be used by the two devices, and to acquire the specifics of how that algorithm is to be employed. The device initiating the negotiation sends a Configure-Request with one option for each of the compression algorithms it supports. The other device compares this list of options to the algorithms it understands. It also checks for any specific details relevant to the option to see if it agrees on how that algorithm should be used. It then sends back the appropriate reply (Ack, Nak, or Reject), and a negotiation ensues until the two devices come up with a common algorithm that both understand. If so, compression is turned on; otherwise, it is not enabled.

The CCP configuration options begin with a Type value that indicates the compression algorithm. When the Type value is 0, this indicates that the option contains information about a special, proprietary compression algorithm that isn't covered by any RFC standards. This information can be used if both devices understand it. Values from 1 to 254 indicate compression algorithms that have been defined for use with CCP. Table 11-1 shows the most common values of the Type field, including the compression algorithm each corresponds to and the number of the RFC that defines it.

**Table 11-1:** PPP Compression Control Protocol (CCP) Compression Algorithms

<b>CCP Option Type Value</b>	<b>Defining RFC</b>	<b>Compression Algorithm (As Given in RFC Title)</b>
0	—	Proprietary
1 and 2	1978	PPP Predictor Compression Protocol
17	1974	PPP Stac LZS Compression Protocol
18	2118	Microsoft Point-to-Point Compression (MPPC) Protocol
19	1993	PPP Gandalf FZA Compression Protocol
21	1977	PPP BSD Compression Protocol
23	1967	PPP LZS-DCP Compression Protocol (LZS-DCP)
26	1979	PPP Deflate Protocol

## ***Compression Algorithm Operation: Compressing and Decompressing Data***

Once an algorithm has been successfully negotiated, the compression algorithm is used to compress data before transmission and to decompress it once received. To compress, the transmitting device takes the data that would normally be put in the Information field of an uncompressed PPP frame and runs it through the compression algorithm. To indicate that a frame has been compressed, the special value

0x00FD (hexadecimal) is placed in the PPP Protocol field. When compression is used with multiple links and the links are compressed independently, a different value is used: 0x00FB.

You'll recall that in a regular uncompressed frame, the Protocol field indicates which layer 3 protocol the data comes from. Since you still need to know this, the original Protocol value is actually prepended to the data before compression. When the data is decompressed, this value is used to restore the original Protocol field, so the receiving device knows to which higher layer the data belongs.

For example, if you use IPCP to encapsulate IP data in PPP, the uncompressed frame would have a value of 0x8021 in the Protocol field. This value (0x8021) would be placed at the start of the data to be compressed. The compressed data would be put in a PPP frame with a Protocol value of 0x00FD. The receiving device would see the value 0x00FD in the Protocol field, recognize the frame as compressed, decompress it, and restore the original frame with 0x8021 as the Protocol value. The discussion of the PPP general frame format in Chapter 12 covers this in more detail.

In theory, a compression algorithm can put more than one PPP data frame into a compressed PPP data frame. Despite this, many, if not most, of the algorithms maintain a one-to-one correspondence, putting each PPP data frame into one compressed frame. Note that LCP frames are not compressed, nor are the control frames used for other protocols. For example, a data frame carrying IP traffic would be compressed, but a control frame for IPCP (the NCP for IP) would not be.

Compression can be combined with encryption. In this case, compression is done before encryption.

**NOTE** *The compression performed by CCP has nothing to do with the header compression options that can be negotiated as part of LCP. That type of compression doesn't involve compressing a data stream using a compression algorithm, but rather a simple way of saving space in headers when both ends of a link agree to do so.*

## PPP Encryption Control Protocol (ECP) and Encryption Algorithms

The PPP authentication protocols Password Authentication Protocol (PAP) and Challenge Handshake Authentication Protocol (CHAP) can be used to ensure that only authorized devices can establish a PPP connection. Once that is done, PPP normally provides no other security to the data being transmitted. In particular, all data is normally sent in the clear (unencrypted), thereby making it easy for someone who intercepts it to read.

For important data that must be kept secure, encryption prior to transmission is a good idea. This can be done at higher layers using something like IPsec, but PPP also provides an optional feature that allows data to be encrypted and decrypted at the data link layer itself using two protocol components:

**PPP Encryption Control Protocol (ECP)** This protocol is responsible for negotiating and managing the use of encryption on a PPP link.

**PPP Encryption Algorithms** A family of encryption algorithms that perform the actual encryption and decryption of data. Several of these are defined in RFCs, and two devices can also negotiate a proprietary encryption method if they want to use one that isn't defined by a public standard.

**KEY CONCEPT** PPP includes an optional encryption feature that provides privacy for data transported over PPP. A number of encryption algorithms are supported. To enable encryption, both devices on a PPP link use the *PPP Encryption Control Protocol (ECP)* to negotiate which algorithm to use. The selected algorithm is then used to encrypt and decrypt PPP data frames.

## **ECP Operation: Encryption Setup**

ECP is usually the only part mentioned when encryption in PPP is discussed, but it is actually used only to configure and control the use of encryption; the algorithms do the real work. This technique allows each implementation to choose which type of encryption it wishes to use.

The original ECP defined only a single encryption method, and a couple of others have since been added. Like CCP, ECP is analogous to the NCPs that negotiate parameters specific to a network layer protocol sent on the link, but it deals with how devices encrypt data, rather than how they transport layer 3 traffic. This also means that like the NCPs, ECP is a light version of LCP and works in the same basic way. Once an ECP link is negotiated, devices can send encrypted frames between each other. When no longer needed, the ECP link can be terminated.

ECP uses the same subset of seven LCP message types that the NCPs use, and it adds two more. The use of these messages for each of the life stages of an ECP link is as follows:

**Link Configuration** Like the NCPs (and also like CCP, of course), encryption configuration is done once ECP reaches the *network layer protocol* phase. The process of setting up encryption and negotiating parameters is accomplished using Configure-Request, Configure-Ack, Configure-Nak, and Configure-Reject messages, as I explained in the description of LCP in Chapter 10, except the configuration options are particular to ECP.

**Link Maintenance** Code-Reject messages can be sent to indicate invalid code values in ECP frames. The two new message types are Reset-Request and Reset-Ack, which are used to reset the encryption (the ECP link) in the event of a detected failure in decryption.

**Link Termination** An ECP link can be terminated using Terminate-Request and Terminate-Ack. Again, remember that like the NCP links, the ECP link is set up within an LCP link, so closing it doesn't terminate the LCP link.

## **ECP Configuration Options and Encryption Algorithms**

ECP configuration options are used solely to negotiate the type of encryption algorithm that will be used by the two devices and the specifics of how that algorithm will be employed. The device initiating the negotiation sends a Configure-Request

with one option for each of the encryption algorithms it supports. The other device compares this list of options to the algorithms it understands. It also checks for any details relevant to the option to see if it agrees on how that algorithm should be used. It then sends back the appropriate reply (Ack, Nak, or Reject), and a negotiation ensues until the two devices come up with a common algorithm that they both understand. If so, encryption is enabled; otherwise, it is turned off.

The ECP configuration options begin with a Type value that indicates the encryption algorithm. When the Type value is 0, this indicates that the option contains information about a special, proprietary encryption method that isn't covered by any RFC standards, which can be used if both devices understand it. Values in the range from 1 to 254 indicate encryption algorithms that have been defined for use with ECP; at present, only two are defined. Table 11-2 shows the values of the Type field, including the encryption algorithm each corresponds to and the number of the RFC that defines it.

**Table 11-2:** PPP Encryption Control Protocol (ECP) Compression Algorithms

ECP Option Type Value	Defining RFC	Encryption Algorithm (As Given in RFC Title)
0	—	Proprietary
2	2420	The PPP Triple-DES Encryption Protocol (3DESE)
3	2419	The PPP DES Encryption Protocol, Version 2 (DESE-bis)

**NOTE** Type value 1 was for the original DES algorithm, which was defined in RFC 1969. It was superseded by DES version 2 in RFC 2419.

### **Encryption Algorithm Operation: Encrypting and Decrypting Data**

Once an encryption algorithm has been successfully negotiated, it is used to encrypt data before transmission and to decrypt data that has been received. To encrypt, the transmitting device takes the data that would normally be put in the Information field of an unencrypted PPP frame and runs it through the encryption algorithm. To indicate that a frame has been encrypted, the special value 0x0053 (hexadecimal) is placed in the PPP Protocol field. When encryption is used with multiple links and the links are encrypted independently, a different value is used: 0x0055.

You'll recall that in a regular unencrypted frame, the Protocol field indicates which layer 3 protocol the data comes from. Since you still need to know this, the original Protocol value is actually prepended to the data before encryption. When the data is decrypted, this value is used to restore the original Protocol field, so the receiving device knows which higher layer the data belongs to.

For example, if you use IPCP to encapsulate IP data in PPP, the unencrypted frame would have a value of 0x8021 (hex) in the Protocol field. This value (0x8021) would be placed at the start of the data to be encrypted. The encrypted data would be put in a PPP frame with a Protocol value of 0x0053. The receiving device would

see the value 0x0053 in the Protocol field, recognize the frame as encrypted, decrypt it, and restore the original frame with 0x8021 as the Protocol value. The discussion of the PPP general frame format in Chapter 12 covers this more completely.

Each encrypted PPP data frame carries exactly one PPP data frame. Note that, unlike what you saw in compression, LCP frames and the control frames used for other protocols *can* be encrypted. Compression can be combined with encryption; in this case, compression is done before encryption.

## PPP Multilink Protocol (MP, MLP, MLPPP)

Most of the time, there is only a single physical layer link between two devices. However, there are some situations for which there may actually be two layer 1 connections between the same pair of devices. This may seem strange. Why would there be more than one link between any pair of machines?

There are a number of situations in which this can occur. A common one is when two links are intentionally placed between a pair of devices. This is often done to increase performance by widening the pipe between two devices, without going to a newer, more expensive technology. For example, if two machines are connected to each other using a regular analog modem that's too slow, a relatively simple solution is to use two analog modem pairs connecting the machines to double bandwidth.

A slightly different situation occurs when multiplexing creates the equivalent of several physical layer channels between two devices, even if they have only one hardware link between them. Consider ISDN, for example. The most common form of ISDN service (ISDN basic rate interface or BRI) creates two 64,000 bps *B channels* between a pair of devices. These B channels are time division multiplexed and carried along with a D channel on a single pair of copper wire, but to the devices, they appear *as if* there were two physical layer links between devices, each of which carries 64 Kbps of data. And the ISDN primary rate interface (PRI) actually creates 23 or more channels, all between the same pair of hardware devices.

In a situation where you have multiple links, you could just establish PPP over each connection independently. However, this is far from an ideal solution, because you would then have to manually distribute the traffic over the two (or more) channels or links that connect them. If you wanted to connect to the Internet, you would need to make separate connections and then choose which one to use for each action. That isn't exactly a recipe for fun, and what's worse is that you could never use all the bandwidth for a single purpose, such as downloading the latest 100 MB Microsoft security patch.

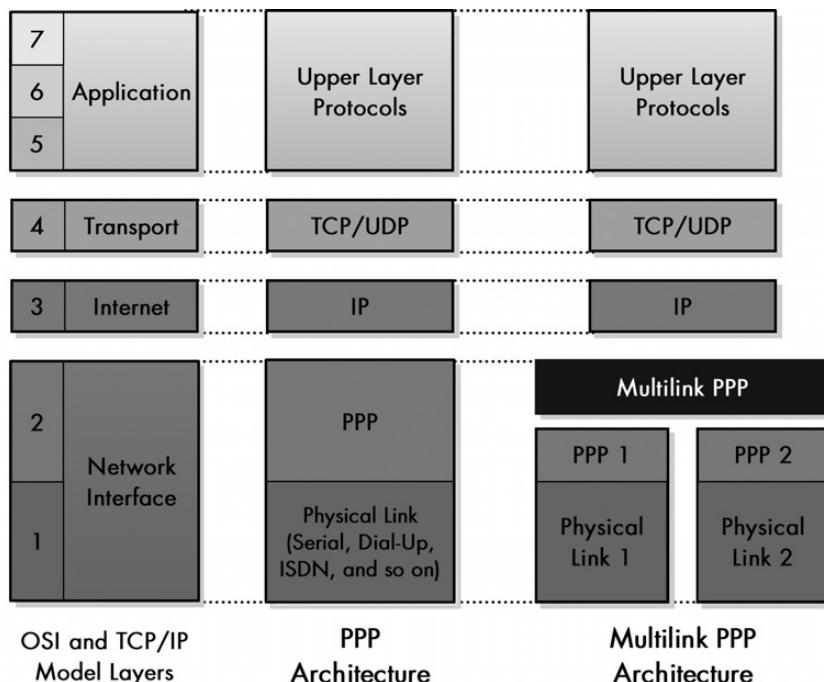
What you really want is a solution that will let you combine multiple links and use them as if they were one high-performance link. Some hardware devices actually allow this to be done at the hardware level itself. In ISDN, this technology is sometimes called *bonding* when done at layer 1. For those hardware units that don't provide this capability, PPP makes it available in the form of the PPP Multilink Protocol (MP). This protocol was originally described in RFC 1717 and was updated in RFC 1990.

**NOTE** The PPP Multilink Protocol is properly abbreviated MP, but it is common to see any of a multitude of other abbreviations used for it. Many of these are actually derived from changing the order of the words in the name into Multilink PPP, so you will frequently see this called ML PPP, MLPPP, MPPP, MLP, and so forth. These are technically incorrect, but widely used, especially MLPPP. I use the correct abbreviation in this book.

## PPP Multilink Protocol Architecture

MP is an optional feature of PPP, so it must be designed to integrate seamlessly into regular PPP operation. To accomplish this, MP is implemented as a new architectural sublayer within PPP. In essence, an MP sublayer is inserted between the regular PPP mechanism and any network layer protocols using PPP, as shown in Figure 11-1. This allows MP to take all network layer data to be sent over the PPP link and spread it over multiple physical connections, without causing either the normal PPP mechanisms or the network layer protocol interfaces to PPP to break.

The column on the left in Figure 11-1 shows the TCP/IP model architecture with corresponding OSI Reference Model layer numbers. The center column shows the normal PPP layer architecture. When MP is used, there are separate PPP implementations running over each of two or more physical links. MP sits, architecturally, between these links and any network layer protocols that will be transported over those links. (In this diagram, only IP is shown because it is most common, but MP can work with multiple network layer protocols, each of which are being sent over each physical link.)



**Figure 11-1: Multilink PPP architecture** When Multilink PPP is used to combine two or more physical links, it sits architecturally above the PPP layers that operate on each physical link.

**KEY CONCEPT** The *PPP Multilink Protocol (MP)* allows PPP to bundle multiple physical links and use them like a single, high-capacity link. It must be enabled during link configuration. Once operational, it works by fragmenting whole PPP frames and sending the fragments over different physical links.

## **PPP Multilink Protocol Setup and Configuration**

To use MP, both devices must have it implemented as part of their PPP software and must negotiate its use. This is done by LCP as part of the negotiation of basic link parameters in the *link establishment* phase (just like LQR, as described earlier in this chapter). Three new configuration options are defined to be negotiated to enable MP:

**Multilink Maximum Received Reconstructed Unit** Provides the basic indication that the device starting the negotiation supports MP and wants to use it. The option contains a value specifying the maximum size of the PPP frame it supports. If the device receiving this option does not support MP, it must respond with a Configure-Reject LCP message.

**Multilink Short Sequence Number Header Format** Allows devices to negotiate the use of a shorter sequence number field for MP frames, for efficiency. (See the section on MP frames in Chapter 12 for a full discussion.)

**Endpoint Discriminator** Uniquely identifies the system. It is used to allow devices to determine which links go to which other devices.

Before MP can be used, a successful negotiation of at least the Multilink Maximum Received Reconstructed Unit option must be performed on each of the links between the two devices. Once this is done and an LCP link exists for each of the physical links, a virtual *bundle* is made of the LCP links, and MP is enabled.

## **PPP Multilink Protocol Operation**

As mentioned previously, MP basically sits between the network layer and the regular PPP links and acts as a middleman. Here is what it does for each direction of communication:

**Transmission** MP accepts datagrams received from any of the network layer protocols configured using appropriate NCPs. It first encapsulates them into a modified version of the regular PPP frame, and then takes that frame and decides how to transmit it over the multiple physical links. Typically, this is done by dividing the frame into *fragments* that are evenly spread out over the set of links. These are then encapsulated and sent over the physical links. However, you can also implement an alternative strategy as well, such as alternating full-sized frames between the links. Also, smaller frames typically aren't fragmented, and neither are control frames such as the ones used for link configuration.

**Reception** MP takes the fragments received from all physical links and reassembles them into the original PPP frame. That frame is then processed like any PPP frame by looking at its Protocol field and passing it to the appropriate network layer protocol.

The fragments used in MP are similar in concept to IP fragments, but of course these are different protocols running at different layers. To PPP or MP, an IP fragment is just an IP datagram like any other.

The fragmenting of data in MP introduces a number of complexities that the protocol must handle. For example, since fragments are being sent roughly concurrently, you need to identify them with a sequence number to facilitate reassembly. You also need some control information to identify the first and last fragments. A special frame format is used for MP fragments to carry this extra information. I describe this in Chapter 12, which also contains more information about how fragmenting is accomplished, as well as an illustration that demonstrates how it works.

## **PPP Bandwidth Allocation Protocol (BAP) and Bandwidth Allocation Control Protocol (BACP)**

The PPP MP allows multiple links between a pair of devices, whether physical or in the form of virtual channels, to be combined into a fat pipe (high-capacity channel). This offers tremendous advantages to many PPP users, because it lets them make optimal use of all their bandwidth, especially for applications such as Internet connectivity. It's no surprise, then, that MP has become one of the most popular features of PPP.

The original standard defining MP basically assumed that multiple links would be combined into a single bundle. For example, if you had two modem links, they would both be connected and then combined, or two B channels in an ISDN link would be combined. After MP was set up, the bundle would be available for either device to use in its entirety.

There's one drawback to this system: The fat pipe is always enabled, and in many cases, it is expensive to have this set up all the time. It often costs more to connect two or more layer 1 links than a single one, and it's not always needed. For example, some ISDN services charge per minute for calls on either of the B channels. In the case of modem dial-up, there are per-minute charges in some parts of the world. Even where regular phone calls are free, there is a cost in the form of tying up a phone line. Consider that in many applications, the amount of bandwidth needed varies over time.

It would be better if you could set up MP so that it could dynamically add links to the bundle when needed (such as when you decided to download some large files), and then automatically drop them when no longer required. This enhancement to the basic MP package was provided in the form of a pair of new protocols described in RFC 2125:

**Bandwidth Allocation Protocol (BAP)** Describes a mechanism where either device communicating over an MP bundle of layer 1 links may request that a link be added to the bundle or removed from it.

**Bandwidth Allocation Control Protocol (BACP)** Allows devices to configure how they want to use BAP.

**KEY CONCEPT** BAP and BACP are used to provide dynamic control over how PPP MP functions.

### **BACP Operation: Configuring the Use of BAP**

Let's start with BACP, since it is the protocol used for the initial setup of the feature. BACP is very similar conceptually to all those other PPP protocols with "Control" in their names, such as LCP, the NCP family, CCP, and ECP, but is actually even simpler. It is used only during link configuration to set up BAP. This is done using Configure-Request, Configure-Ack, Configure-Nak, and Configure-Reject messages, just as described in the LCP topic.

The only configuration option that is negotiated in BACP is one called Favored-Peer, which is used to ensure that a problem does not occur if the two devices on the link try to send the same request at the same time. If both devices support BAP, then the BACP negotiation will succeed and BAP will be activated.

### **BAP Operation: Adding and Removing Links**

BAP defines a set of messages that can be sent between devices to add or drop links to and from the current PPP bundle. What's particularly interesting about BAP is that it includes the tools necessary to have a device actually initiate different types of physical layer connections (such as dialing a modem for bundled analog links or enabling an extra ISDN channel) when more bandwidth is required. It then shuts them down when they're no longer needed.

Here's a brief description of the BAP message types:

**Call-Request and Call-Response** When one device on the link wants to add a link to the bundle and initiate the new physical layer link itself, it sends a Call-Request frame to tell the other device, which replies with a Call-Response.

**Callback-Request and Callback-Response** These are just like the two previous message types, except that they're used when a device wants its peer (the other device on the link) to initiate the call to add a new link. So, if Device A says, "I need more bandwidth but I want you to call me, instead of me calling you," it sends Device B a Callback-Request.

**Call-Status-Indication and Call-Status-Response** After a device attempts to add a new link to the bundle (after sending a Call-Request or receiving a Callback-Request), it reports the status of the new link using the Call-Status-Indication frame. The other device then replies with a Call-Status-Response.

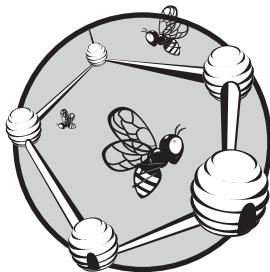
**Link-Drop-Query-Request and Link-Drop-Query-Response** One device uses these messages to request that a link be dropped, and the other uses them to respond to that request.

Note that the decision of when to add or remove links is not made by these protocols. It is left up to the particular implementation.



# 12

## PPP PROTOCOL FRAME FORMATS



The Point-to-Point Protocol (PPP) protocol suite includes a number of different protocols used to send both data and control information in different ways. Each of these packages information into messages called *frames*, each of which follows a particular *frame format*. PPP starts with a general frame format that encompasses all frames sent on the link and then includes more specific formats for different purposes. Understanding these formats not only makes diagnosing PPP issues easier, but also helps make more clear how the key PPP protocols function.

In this chapter, I illustrate the most common frame formats used for sending both data and control information over PPP. I begin with an explanation of the overall format used for all PPP frames. I also describe the general format used for the various control protocols and the option format that most of them use. (One of the nice things about PPP is that so many of the protocols use control frames with a common format.)

I then specifically list the frames used for Link Control Protocol (LCP) and the authentication protocols (PAP and CHAP). I also describe the special format used by the PPP Multilink Protocol (MP) to transport fragments of data over bundled links.

**NOTE** Due to the sheer number of different protocols in PPP (dozens) and the fact that many have their own unique options, I won't describe all the specific frame formats and option formats for every protocol in detail here. Please refer to the appropriate RFCs (listed in Chapter 9 for more detail).

## PPP General Frame Format

All messages sent using PPP can be considered either *data* or *control information*. The word *data* describes the higher-layer datagrams you are trying to transport here at layer 2. This is what our “customers” are giving us to send. Control information is used to manage the operation of the various protocols within PPP itself. Even though different protocols in the PPP suite use many types of frames, at the highest level, they all fit into a single, *general* frame format.

You'll recall that the basic operation of the PPP suite is based on the ISO High-Level Data Link Control (HDLC) protocol. This becomes very apparent when you look at the structure of PPP frames overall—they use the same basic format as HDLC, even to the point of including certain fields that aren't strictly necessary for PPP itself. The only major change is the addition of a new field to specify the protocol of the encapsulated data. The general structure of PPP frames is defined in RFC 1662, a companion to the main PPP standard RFC 1661.

The general frame format for PPP, showing how the HDLC framing is applied to PPP, is described in Table 12-1 and illustrated in Figure 12-1.

**Table 12-1:** PPP General Frame Format

Field Name	Size (Bytes)	Description
Flag	1	Indicates the start of a PPP frame. Always has the value 01111110 binary (0x7E hexadecimal, or 126 decimal).
Address	1	In HDLC this is the address of the destination of the frame. But in PPP you are dealing with a direct link between two devices, so this field has no real meaning. It is thus always set to 11111111 (0xFF or 255 decimal), which is equivalent to a broadcast (it means “all stations”).
Control	1	This field is used in HDLC for various control purposes, but in PPP it is set to 00000011 (3 decimal).
Protocol	2	Identifies the protocol of the datagram encapsulated in the Information field of the frame. See the “Protocol Field Ranges” section for more information on the Protocol field.
Information	Variable	Zero or more bytes of payload that contain either data or control information, depending on the frame type. For regular PPP data frames, the network layer datagram is encapsulated here. For control frames, the control information fields are placed here instead.
Padding	Variable	In some cases, additional dummy bytes may be added to pad out the size of the PPP frame.
Frame Check Sequence	2 (or 4)	A checksum computed over the frame to provide basic protection against errors in transmission. This is a CRC similar to the one used for other layer 2 protocol error-protection schemes such as the one used in Ethernet. It can be either 16 bits or 32 bits in size (the default is 16 bits). The FCS is calculated over the Address, Control, Protocol, Information, and Padding fields.
Flag	1	Indicates the end of a PPP frame. Always has the value 01111110 binary (0x7E hexadecimal, or 126 decimal).

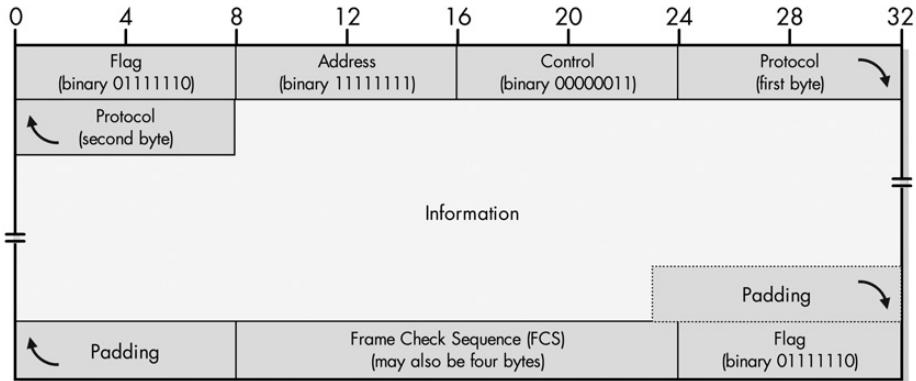


Figure 12-1: PPP general frame format

### Protocol Field Ranges

The Protocol field is the main frame type indicator for the device receiving the frame. For data frames, this is normally the network layer protocol that created the datagram; for control frames, it is usually the PPP protocol that created the control message. In the case of protocols that modify data such as when compression (CCP) or encryption (ECP) are used (as explained in the previous chapter), this field identifies the data as being either compressed or encrypted, and the original Protocol value is extracted after the Information field is decompressed/decrypted.

All PPP frames are built on the general format shown in Figure 12-1. The first three bytes are fixed in value, followed by a two-byte Protocol field that indicates the frame type. The variable-length Information field is formatted in a variety of ways, depending on the PPP frame type. Padding may be applied to the frame, which concludes with an FCS field of either two or four bytes (two bytes shown here) and a trailing Flag value of 0x7E. (See Figure 12-2 for an example of how this format is applied.)

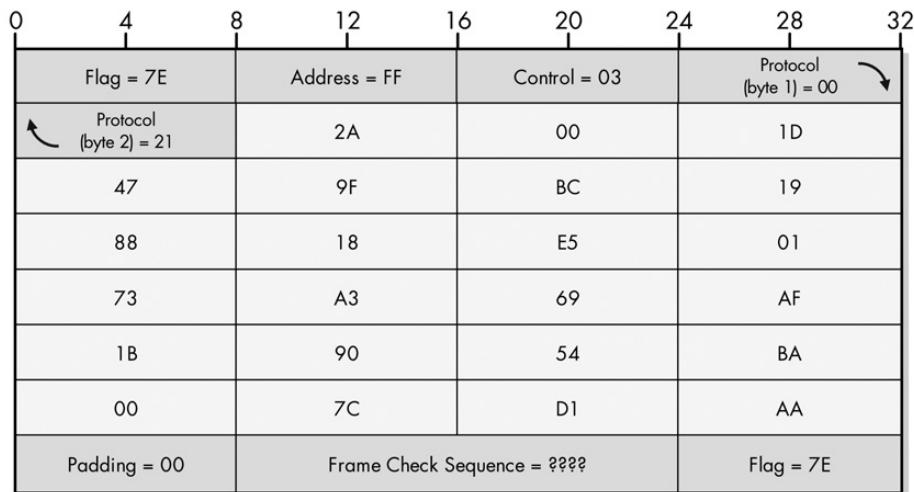
There are dozens of network layer protocols and PPP control protocols, and a correspondingly large number of Protocol values. The main PPP standard defines four ranges for organizing these values, as shown in Table 12-2.

The standard also specifies that the Protocol value must be assigned so that the first octet is even and the second octet is odd. So, for example, 0x0021 is a valid value, but 0x0121 and 0x0120 are not. (The reason for this will become apparent shortly.) There are also certain blocks that are reserved and not used.

Figure 12-2 shows one common application of the PPP general frame format: carrying data. The value 0x0021 in the *Protocol* field marks this as an IPv4 datagram. This sample has 1 byte of *Padding* and a 2-byte *FCS* as well. (Obviously real IP datagrams are longer than the 32 bytes shown here! These bytes are arbitrary and don't represent a real datagram.) See Figure 12-12 for an illustration of how this same data frame is formatted and then fragmented for transmission over multiple links using the PPP Multilink Protocol (MP).

**Table 12-2: PPP Protocol Field Ranges**

Protocol Field Range (Hexadecimal)	Description
<b>0000–3FFF</b>	Encapsulated network layer datagrams that have an associated NCP (see Chapter 10). In this case, control frames from the corresponding NCP use a Protocol field value that is computed by adding 8 to the first octet of the network layer Protocol value. For example, for IP the Protocol value is 0021, and control frames from the IP Control Protocol (IPCP) use Protocol value 8021. This range also includes several values used for specially processed encapsulated datagrams, such as when compression or encryption is employed.
<b>4000–7FFF</b>	Encapsulated datagrams from “low-volume” protocols. These are protocols that do not have an associated NCP.
<b>8000–BFFF</b>	NCP control frames that correspond to the network layer Protocol values in the 0000–3FFF range.
<b>C000–FFFF</b>	Control frames used by LCP and LCP support protocols such as PAP and CHAP. Some miscellaneous protocol values are included here as well.



**Figure 12-2: Sample PPP data frame** An example of a PPP data frame containing an abbreviated 23-byte IP datagram.

### Protocol Field Values

The full list of PPP Protocol values is maintained by the Internet Assigned Numbers Authority (IANA), along with all the other different reserved numbers for Internet standards. Table 12-3 shows some of the more common values.

**Table 12-3:** Common Protocols Carried in PPP Frames and Protocol Field Values

Protocol Type	Protocol Field Value (Hex)	Protocol
Encapsulated Network Layer Datagrams	0021	Internet Protocol version 4 (IPv4)
	0023	OSI Network Layer
	0029	AppleTalk
	002B	Novell Internetworking Packet Exchange (IPX)
	003D	PPP Multilink Protocol (MP) fragment
	003F	NetBIOS Frames (NBF/NetBEUI)
	004D	IBM Systems Network Architecture (SNA)
	0053	Encrypted Data (using ECP and a PPP encryption algorithm)
	0055	Individual Link Encrypted Data under PPP Multilink
	0057	Internet Protocol version 6 (IPv6)
Low-Volume Encapsulated Protocols	00FB	Individual Link Compressed Data under PPP Multilink
	00FD	Compressed Data (using CCP and a PPP compression algorithm)
Network Control Protocol (NCP) Control Frames	4003	CDPD Mobile Network Registration Protocol
	4025	Fibre Channel
	8021	PPP Internet Protocol Control Protocol
	8023	PPP OSI Network Layer Control Protocol
	8029	PPP AppleTalk Control Protocol
	802B	PPP IPX Control Protocol
	803F	PPP NetBIOS Frames Control Protocol
LCP and Other Control Frames	804D	PPP SNA Control Protocol
	8057	PPP IPv6 Control Protocol
	C021	PPP Link Control Protocol (LCP)
	C023	PPP Password Authentication Protocol (PAP)
	C025	PPP Link Quality Report (LQR)
	C02B	PPP Bandwidth Allocation Control Protocol (BACP)
	C02D	PPP Bandwidth Allocation Protocol (BAP)
	C223	PPP Challenge Handshake Authentication Protocol (CHAP)

### **PPP Field Compression**

PPP uses the HDLC basic framing structure, which includes two fields that are needed in HDLC but aren't in PPP due to how the latter operates. The fields are the Address and Control fields. Why bother sending two bytes that have the same value for every frame and aren't used for anything? Originally, they were maintained for compatibility, but this reduces efficiency.

To avoid wasting two bytes in every frame, it is possible during initial link setup using the Link Control Protocol (LCP) for the two devices on the link to negotiate a feature called *Address and Control Field Compression* (ACFC) using the LCP option by that same name. When enabled, this feature simply causes these two fields not to be sent for most PPP frames (but not for LCP control frames). In fact, the feature would be better named *Address and Control Field Suppression*, because the fields are just suppressed and compressed down to nothing.

Even when devices agree to use field compression, they must still be capable of receiving both compressed and uncompressed frames. They differentiate one from the other by looking at the first two bytes after the initial Flag field. If they contain the value 0xFF03, they must be the Address and Control fields; otherwise, those fields were suppressed. (The value 0xFF03 is not a valid Protocol field value, so there is no chance of ambiguity.)

Similarly, it is also possible for the two devices on the link to negotiate compression of the Protocol field, so it takes only one byte instead of two. This is done generally by dropping the first byte if it is zero, a process called *Protocol Field Compression* (PFC). Recall that the first byte must be even and the second odd. Thus, a receiving device examines the evenness of the first byte of the Protocol field in each frame. If it is odd, this means that a leading byte of zeros in the Protocol field has been suppressed, because the first byte of a full two-byte Protocol value must be even.

**NOTE** *This field compression (really suppression) has nothing to do with data compression using PPP's Compression Control Protocol (CCP) and compression algorithms.*

## PPP General Control Protocol Frame Format and Option Format

The general frame format you just saw is used for all of the many frame types defined in the PPP protocol suite. Within that format, the Information field carries either encapsulated layer 3 user data or encapsulated control messages. These control messages contain specific information that is used to configure, manage, and discontinue PPP links, and to implement the various features that comprise PPP.

There are many different PPP control protocols that usually can be distinguished by the word *Control* appearing in their names. These include the main PPP Link Control Protocol (LCP); a family of Network Control Protocols (NCPs) such as IPCP, IPXCP, and so forth; and also control protocols for implementing features, such as the Compression Control Protocol (CCP) and the Encryption Control Protocol (ECP). The authentication protocols Password Authentication Protocol (PAP) and Challenge Handshake Authentication Protocol (CHAP) lack *Control* in the name but also fall into this category.

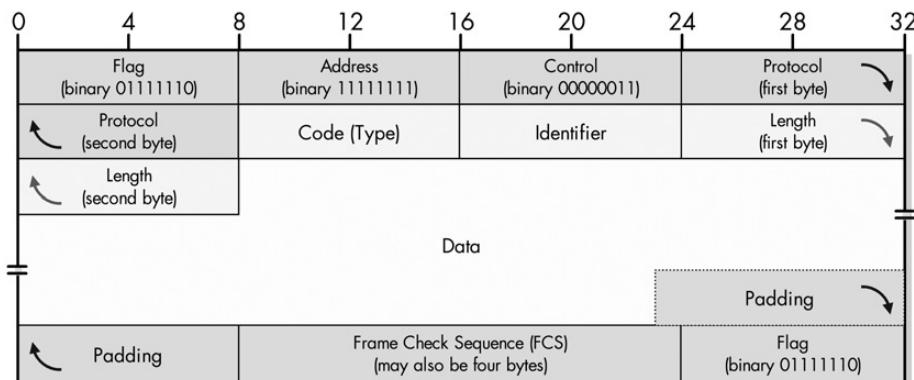
The control protocols each use control messages in a slightly different way, but there is also a great deal of commonality between the messages. This is because, as I explained in my discussions of the PPP protocols, most of the control protocols—such as the NCP family, CCP, and ECP—are implemented as subsets of the functionality of the LCP. They perform many of the same functions, so the PPP designers wisely adapted the LCP messaging system for these other control protocols.

This all means that control protocol frames have a common format that fits within the overall general frame format in PPP. Even protocols like PAP and CHAP, which aren't based on LCP, use this general control frame format, which is described in Table 12-4.

**Table 12-4: PPP Control Message Format**

Field Name	Size (Bytes)	Description
Code (Type)	1	A single byte value that indicates what type of control message is in this control frame. It is sometimes instead called Type in certain PPP standards.
Identifier	1	This is a label field that's used to match up requests with replies. When a request is sent, a new Identifier is generated. When a reply is created, the value from the Identifier field in the request that prompted the reply is used for the reply's Identifier field.
Length	2	Specifies the length of the control frame. This is needed because the Data field is variable in length. The Length field is specified in bytes and includes all the fields in the control frame including the Code, Identifier, Length, and Data fields.
Data	Variable	Contains information specific to the message type. The different uses of this field are described later in this chapter.

This entire structure becomes the payload of a PPP frame, meaning that it fits into the Information field of a PPP frame, as shown in Figure 12-3. The four fields of the PPP control message format fit within the Information field of the PPP general frame format. The Data field is subsequently filled in with data specific to the control message type. Thus, the Length field is equal in size to that of the Information field in the PPP frame. The Protocol field of a control frame is set to match the protocol that generated the control frame. For example, it would be 0xC021 for an LCP frame.



**Figure 12-3: PPP control message format**

### PPP Control Messages and Code Values

The Code field indicates the type of control frame within the particular control protocol. Some protocols have a unique set of codes used only by that particular protocol; examples include the authentication protocols (PAP and CHAP) and the

Bandwidth Allocation Protocol (BAP). Since the NCPs and many of the feature control protocols like CCP and ECP are based on LCP, they use a common set of message codes and types. Table 12-5 shows these common message codes and indicates which control protocols use them.

**Table 12-5:** PPP Control Messages, Code Values, and PPP Protocol Usage

Code Value	Control Message	LCP	NCPs	CCP and ECP
1	Configure-Request	✓	✓	✓
2	Configure-Ack	✓	✓	✓
3	Configure-Nak	✓	✓	✓
4	Configure-Reject	✓	✓	✓
5	Terminate-Request	✓	✓	✓
6	Terminate-Ack	✓	✓	✓
7	Code-Reject	✓	✓	✓
8	Protocol-Reject	✓		
9	Echo-Request	✓		
10	Echo-Reply	✓		
11	Discard-Request	✓		
12	Identification	✓		
13	Time-Remaining	✓		
14	Reset-Request			✓
15	Reset-Ack			✓

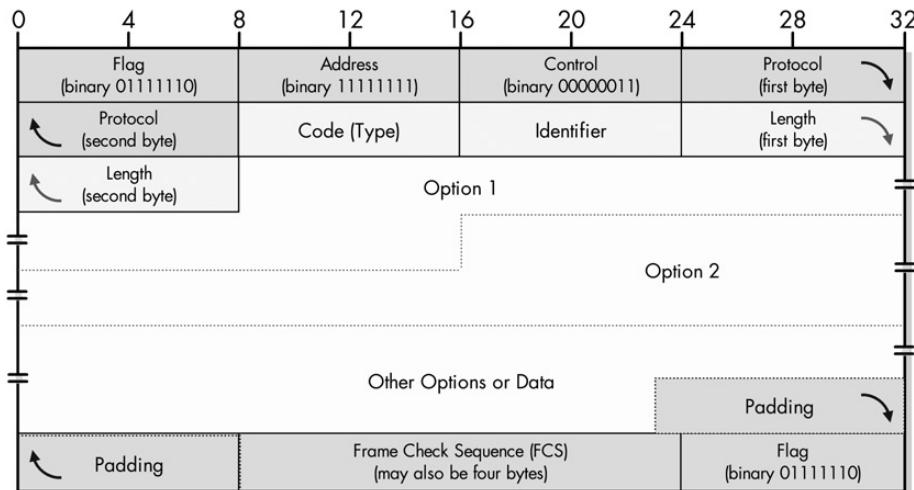
**NOTE** I describe the specific ways these frame types are used in the individual topics on LCP, the NCPs, CCP, and ECP in Chapters 10 and 11.

The contents of the Data field depend entirely on the type of control message. In some cases, no extra data needs to be sent at all, in which case the Data field may be omitted. In other control messages, it carries information relevant to the message type. For example, a *Code-Reject* message carries in the Data field a copy of the frame that was rejected.

### PPP Control Message Option Format

The various *Configure-* messages are used to negotiate configuration options in LCP and the other control protocols. In their Data fields, they carry one or more options that are, again, specific to the protocol using them. For example, LCP uses one set of configuration options for the link as a whole, CCP uses options to negotiate a compression algorithm, MP uses it to set up multilink bundles, and so on. Figure 12-4 shows how these options, which can vary in length, are placed in the Data field of a PPP control message (which is nested inside the general PPP frame format).

This diagram shows a sample PPP control message carrying options in its Data field. Any number of options can be included and mixed with other data, depending on the needs of the message.



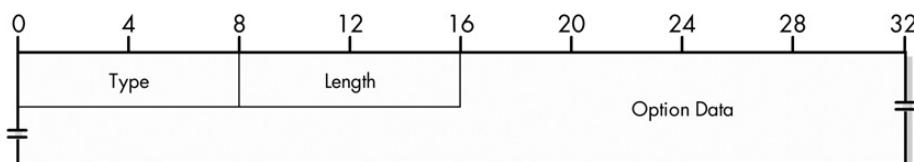
**Figure 12-4:** PPP control message carrying options

Again, there is commonality here. While every option is different, they all use the same basic format. Each option that appears in any of the many PPP control message types consists of the triplet of Type, Length, and Data, as shown in Table 12-6 and illustrated in Figure 12-5.

**Table 12-6:** PPP Control Message Option Format

Field Name	Size (Bytes)	Description
Type	1	A type value that indicates the option type. The set of Type values is unique to each protocol. So, for example, LCP has one set of Type values corresponding to its configuration options, each NCP has a different set, CCP has its own set, and so on.
Length	1	Specifies the length of the option in bytes.
Data	Variable	Contains the specific data for the configuration option.

The configuration options are described briefly in the individual protocol topics. I am not showing the specific contents of each option because there are just too many of them. These are in the RFCs.



**Figure 12-5:** PPP control message option format

## **Summary of PPP Control Message Formatting**

My intention here has been to show you the general format used for the different control protocols because they are so similar and I don't have the time or space to describe each protocol's frames individually. Here's a quick summary:

- The PPP general frame format is used for all frames, including all control frames. Its Information field contains the payload, which carries the entire control message within it for control frames.
- The control frame is structured using the general format I gave at the start of this topic. The Code value indicates the type of control frame for each control protocol. The Data field is variable in length, and contains data for that control frame, which in some cases may include one or more configuration options.
- For configuration control frames like *Configure-Request* and *Configure-Ack*, the Data field contains an encapsulated set of options using the general structure in the second table in this topic. Each option has its own Data subfield that contains data specific to that option.

To help make this more clear, the next two sections provide more specific examples of frame formats for LCP and the authentication protocols.

## **PPP Link Control Protocol (LCP) Frame Formats**

You just explored the general format used by the various protocols in PPP that exchange control messages. Of the many control protocols in PPP, LCP is the most important, because it is responsible for basic PPP link setup and operation. It is also the protocol used as a template for many of the other control protocols.

Since it is so central to PPP, and since many of the other protocols use a similar messaging system, let's make the general frame format (shown in Figure 12-5) more concrete by showing the specific frame formats used for each of the LCP control frames. There are 13 different frame formats, however, and since they have many fields in common I've combined them into a single large summary table. Table 12-7 shows the contents and meaning for each of the fields in the 13 LCP frame types.

**NOTE** *LCP frame types 5, 6, 9, 10, 11, 12, and 13 allow an additional amount of data to be included in the Data field in a manner not strictly described by the protocol. The PPP standard says that there may be zero or more octets that "contain uninterpreted data for use by the sender" and "may consist of any binary value" (RFC 1661). The inclusion of this uninterpreted data is left as an implementation-dependent option.*

All LCP control frames are encapsulated into a PPP frame by placing the frame structure into its Information field, as you saw earlier. The Protocol field is set to 0xC021 for LCP. (For an explanation of how the frames are used, see the operational description of LCP in Chapter 10.)

**Table 12-7: PPP Link Control Protocol (LCP) Frame Types and Fields**

Frame Type	Code Field	Identifier Field	Length Field	Data Field
<b>Configure-Request</b>	1	New value generated for each frame	4 + length of all included configuration options	Configuration options to be negotiated by the two peers on a link. (The previous section in this chapter describes the general format of configuration options.)
<b>Configure-Ack</b>	2	Copied from the Identifier field of the Configure-Request frame for which this Configure-Ack is a reply	4 + length of all included configuration options	Configuration options being positively acknowledged (accepted during negotiation of the link).
<b>Configure-Nak</b>	3	Copied from the Identifier field of the Configure-Request frame for which this Configure-Nak is a reply	4 + length of all included configuration options	Configuration options being negatively acknowledged (renegotiation requested).
<b>Configure-Reject</b>	4	Copied from the Identifier field of the Configure-Request frame for which this Configure-Reject is a reply	4 + length of all included configuration options	Configuration options being rejected (since the device cannot negotiate them).
<b>Terminate-Request</b>	5	New value generated for each frame	4 (or more if extra data is included)	Not required. See note preceding this table.
<b>Terminate-Ack</b>	6	Copied from the Identifier field of the matching Terminate-Request	4 (or more if extra data is included)	Not required. See note preceding this table.
<b>Code-Reject</b>	7	New value generated for each frame	4 + length of rejected frame	A copy of the LCP frame that was rejected. This is not the complete PPP frame; just the LCP control portion from its Information field.
<b>Protocol-Reject</b>	8	New value generated for each frame	6 + length of rejected frame	The first two bytes contain the Protocol value of the frame rejected. The rest contains a copy of the Information field from the frame rejected.
<b>Echo-Request</b>	9	New value generated for each frame	8 (or more if extra data is included)	Contains a four-byte "magic number" used to detect looped-back links, if the appropriate configuration option has been negotiated; otherwise, set to zero. May also contain additional uninterpreted data; see note preceding this table.
<b>Echo-Reply</b>	10	Copied from the Identifier field of the matching Echo-Request	8 (or more if extra data is included)	Contains a four-byte "magic number" used to detect looped-back links, if the appropriate configuration option has been negotiated; otherwise, set to zero. May also contain additional uninterpreted data; see note preceding this table.

(continued)

**Table 12-7: PPP Link Control Protocol (LCP) Frame Types and Fields (continued)**

<b>Frame Type</b>	<b>Code Field</b>	<b>Identifier Field</b>	<b>Length Field</b>	<b>Data Field</b>
<b>Discard-Request</b>	11	New value generated for each frame	8 (or more if extra data is included)	Contains a four-byte “magic number” used to detect looped-back links, if the appropriate configuration option has been negotiated; otherwise, set to zero. May also contain additional uninterpreted data; see note preceding this table.
<b>Identification</b>	12	New value generated for each frame	8 (or more if extra data is included)	Contains a four-byte “magic number” used to detect looped-back links, if the appropriate configuration option has been negotiated; otherwise, set to zero. May also contain additional uninterpreted data; see note preceding this table.
<b>Time-Remaining</b>	13	New value generated for each frame	12 (or more if extra data is included)	Contains a four-byte “magic number” used to detect looped-back links, if the appropriate configuration option has been negotiated; otherwise, set to zero. Also contains a four-byte value indicating the number of seconds remaining in the current session. A value of all ones in this field is interpreted as forever, meaning the session will not expire. May also contain additional uninterpreted data; see note preceding this table.

## PAP and CHAP Frame Formats

For links where security is important, PPP provides two optional authentication protocols, PAP and CHAP. These are used during initial link setup by the LCP to deny PPP connections to unauthorized devices.

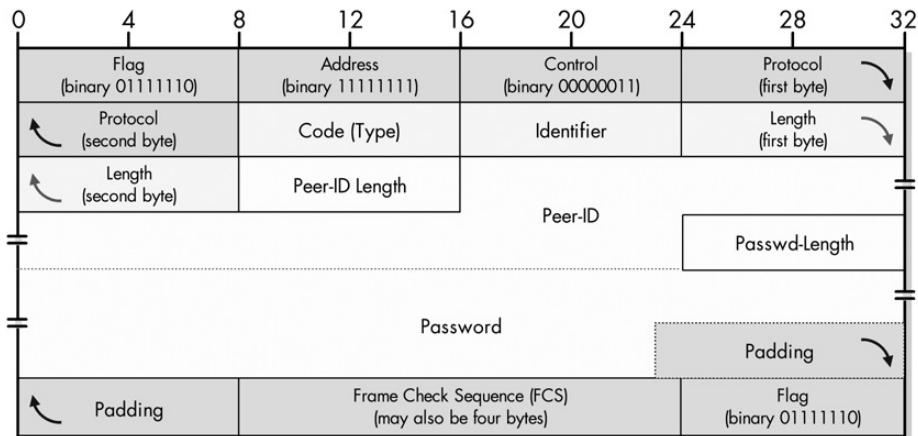
PAP and CHAP are control protocols and thus use the same basic control protocol frame format described earlier in this section. However, since they have a very different purpose than LCP and many of the other control protocols, they use a distinct set of frames with their own unique set of Code values. PAP uses three different control frame types, and CHAP uses four. Let’s look at how PAP and CHAP frames are constructed.

### PPP PAP Control Frame Formats

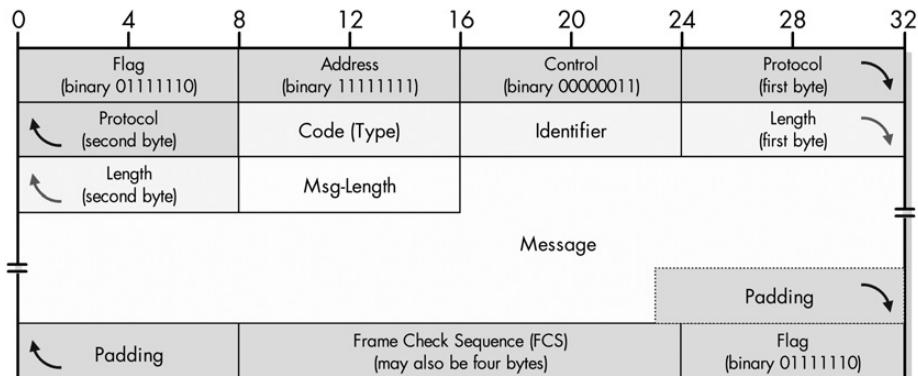
PAP’s three control frames are constructed as shown in Tables 12-8 and 12-9. The *Authenticate-Request* uses one format, as illustrated in Figure 12-6, while the other two frame types use a different format, as shown in Figure 12-7.

**Table 12-8: PPP Password Authentication Protocol (PAP) Frame Formats**

Frame Type	Code Field	Identifier Field	Length Field	Data Field
<b>Authenticate-Request</b>	1	New value generated for each frame	6 + length of Peer-ID + length of password	Contains the user name and password for authentication. This is carried in four subfields and arranged as shown in Table 12-9.
<b>Authenticate-Ack</b>	2	Copied from the Identifier field of the Authenticate-Request frame for which this is a reply	5 + length of included Message	Contains a one-byte Msg-Length subfield that specifies the length of the Message subfield that follows it. The Message subfield contains an arbitrary string of data whose use is implementation-dependent. It may be used to provide an indication of authentication success or failure to the user. If not used, the Msg-Length field is still included, but its value is set to zero.
<b>Authenticate-Nak</b>	3			



**Figure 12-6: PPP PAP Authenticate-Request frame format**



**Figure 12-7: PPP PAP Authenticate-Ack and Authenticate-Nak frame format**

**Table 12-9:** PPP PAP Authenticate-Request Frame Subfields

<b>Subfield Name</b>	<b>Size (Bytes)</b>	<b>Description</b>
Peer-ID Length	1	Length of the Peer-ID field, in bytes
Peer-ID	Variable	Name of the device to be authenticated; equivalent in concept to a user name
Passwd-Length	1	Length of the Password field, in bytes
Password	Variable	Password corresponding to the name being authenticated

### **PPP CHAP Control Frame Formats**

The four CHAP frame types are formatted as shown in Tables 12-10 and 12-11. The Challenge and Response frames use one message format, as illustrated in Figure 12-8, while the Success and Failure frames use a different one, as shown in Figure 12-9.

**Table 12-10:** PPP Challenge Handshake Authentication Protocol (CHAP) Formats

<b>Frame Type</b>	<b>Code Field</b>	<b>Identifier Field</b>	<b>Length Field</b>	<b>Data Field</b>
<b>Challenge</b>	1	New value generated for each frame	5 + length of challenge text + length of Name	Carries the challenge text or response text and a system identifier. This information is carried in three subfields, as shown in Table 12-11.
<b>Response</b>	2	Copied from the Identifier field of the Challenge frame for which this is a reply	5 + length of Value + length of Name	
<b>Success</b>	3	Copied from the Identifier field of the Response frame for which this is a reply	4 (or more if extra data is included)	May contain an arbitrary, implementation-dependent Message field to indicate to the user whether authentication was successful or failed.
<b>Failure</b>	4			

**Table 12-11:** CHAP Challenge and Response Frame Subfields

<b>Subfield Name</b>	<b>Size (Bytes)</b>	<b>Description</b>
Value-Size	1	Length of the Value subfield that follows, in bytes
Value	Variable	For a Challenge frame, contains the challenge text used in the initial challenge; for a Response frame, contains the encrypted challenge text being returned to the authenticator
Name	Variable	One or more bytes of text used to identify the device that sent the frame

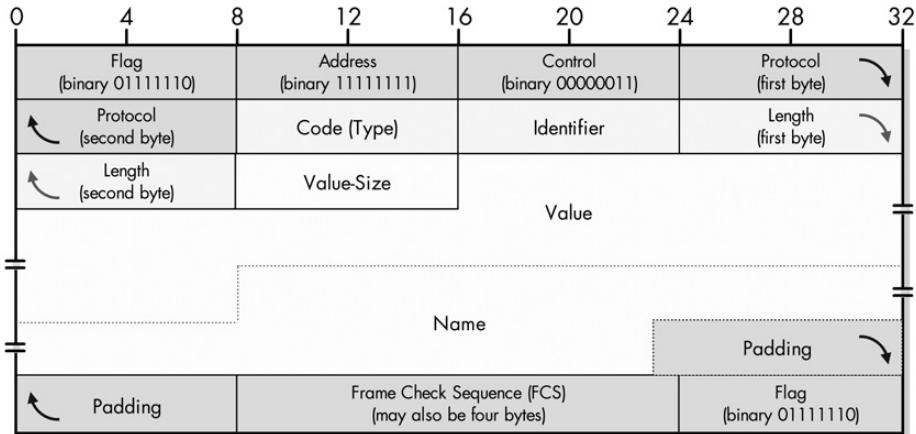


Figure 12-8: PPP CHAP Challenge and Response frame format

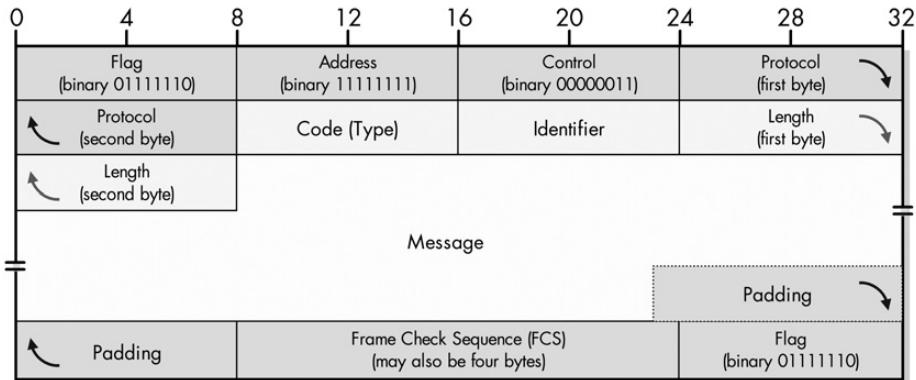


Figure 12-9: PPP CHAP Success and Failure frame format

## PPP Multilink Protocol (MP) Frame Format

Some devices are connected not by a single physical layer link but by two or more. These may be either multiple physical connections, such as two connected pairs of modems, or multiplexed virtual layer 1 connections like ISDN B channels. In either case, PPP MP can be used to aggregate the bandwidth of these physical links to create a single, high-speed *bundle*. I describe how this is done in Chapter 11.

After MP is configured and starts working, it operates by employing a strategy for dividing up regular PPP frames among the many individual physical links that compose the MP bundle. This is usually accomplished by chopping up the PPP frames into pieces called *fragments* and spreading them across the physical links. This allows the traffic on the physical links to be easily balanced.

## **PPP MP Frame Fragmentation Process**

To accomplish this fragmentation process, the device must follow this three-step process:

1. **Original PPP Frame Creation** The data or other information to be sent is first formatted as a whole PPP frame, but in a modified form, as we will see momentarily.
2. **Fragmentation** The full-sized PPP frame is chopped into fragments by MP.
3. **Encapsulation** Each fragment is encapsulated in the Information field of a new PPP MP fragment frame, along with control information that allows the fragments to be reassembled by the recipient.

Several of the fields that normally appear in a whole PPP frame aren't needed if that frame is going to then be divided and placed into other PPP MP frames, so when fragmentation is to occur, they are omitted when the original PPP frame is constructed for efficiency's sake. These are fields that are not used when MP is employed:

- The Flag fields at the start and end are used only for framing for transmission and aren't needed in the logical frame being fragmented.
- The FCS field is not needed, because each fragment has its own FCS field.
- The special compression options that are possible for any PPP frame are used when creating this original frame—that is, the Address and Control Field Compression (APCP) and Protocol Field Compression (PFC). This means that there are no Address or Control fields in the frame, and the Protocol field is only one byte in size. Note that this inherently restricts fragments to carrying only certain types of information.

**KEY CONCEPT** The PPP Multilink Protocol (MP) normally divides data among physical links by creating an original PPP frame with unnecessary headers removed, and then dividing it into fragment frames. Each fragment includes special headers that allow for the reassembly of the original frame by the recipient device.

These changes save a full eight bytes on each PPP frame that will be fragmented. As a result, the original PPP frame has a very small header, consisting of only a one-byte Protocol field. The Protocol value of each fragment is set to 0x003D to indicate a MP fragment, while the Protocol field of the original frame becomes the first byte of data in the first fragment.

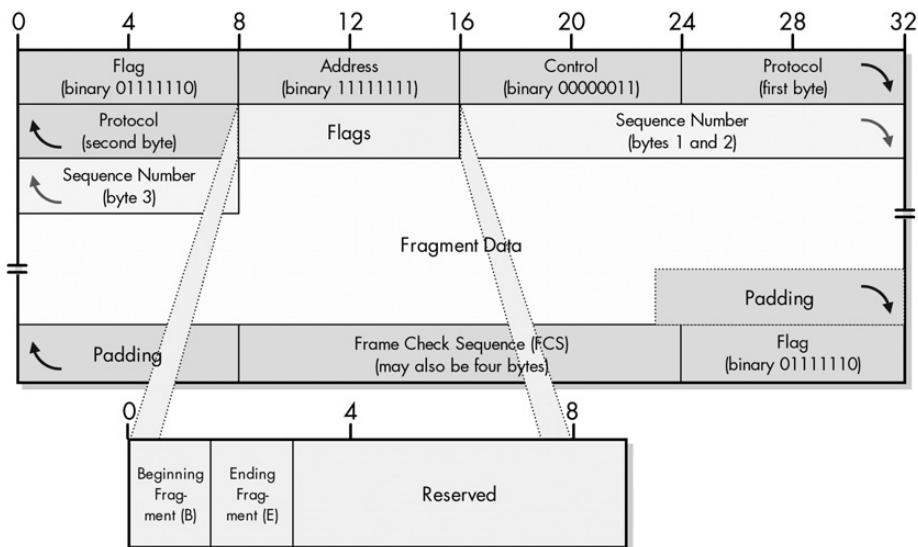
## **PPP MP Fragment Frame Format**

The Information field of each fragment uses a substructure that contains a four-field *MP header* along with one fragment of the original PPP frame, as shown in Table 12-12.

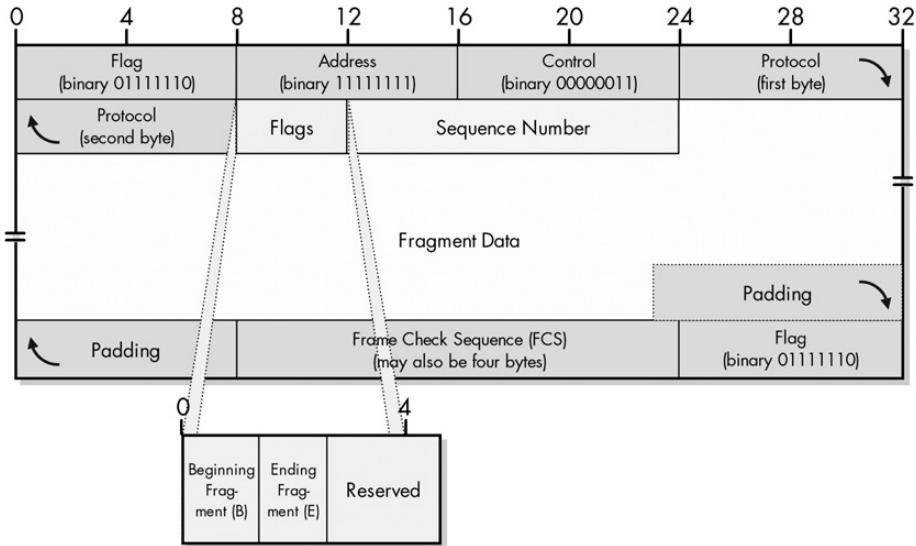
**Table 12-12: PPP Multilink Protocol Fragment Frame Format**

Field Name	Size (Bytes)	Description
B	1/8 (1 bit)	Beginning Fragment Flag: When set to 1, flags this fragment as the first of the split-up PPP frame. It is set to 0 for other fragments.
E	1/8 (1 bit)	Ending Fragment Flag: When set to 1, flags this fragment as the last of the split-up PPP frame. It is set to 0 for other fragments.
Reserved	2/8 (2 bits) or 6/8 (6 bits)	Not used; set to 0.
Sequence Number	1 1/2 (12 bits) or 3 (24 bits)	When a frame is split up, the fragments are given consecutive sequence numbers so the receiving device can properly reassemble them.
Fragment Data	Variable	The actual fragment from the original PPP frame.

As you can see, the MP frame format comes in two versions: the long format uses a four-byte header, while the short format requires only four bytes. The default MP header format uses a 24-bit Sequence Number and has 6 reserved bits, as shown in Figure 12-10. When MP is set up, it is possible for devices to negotiate the Multilink Short Sequence Number Header Format configuration option. If this is done successfully, shorter 12-bit Sequence Numbers are used instead. Four of the reserved bits are also truncated, to save two bytes on each frame, as illustrated in Figure 12-11. (Considering that 12 bits still allows for over 4,000 fragments per PPP frame, this is usually more than enough!)



**Figure 12-10: PPP MP long fragment frame format** The long PPP MP frame format uses a full byte for flags and a 24-bit Sequence Number.



**Figure 12-11: PPP MP short fragment frame format** The short version of the PPP MP format uses 4 bits for flags and a 12-bit Sequence Number.

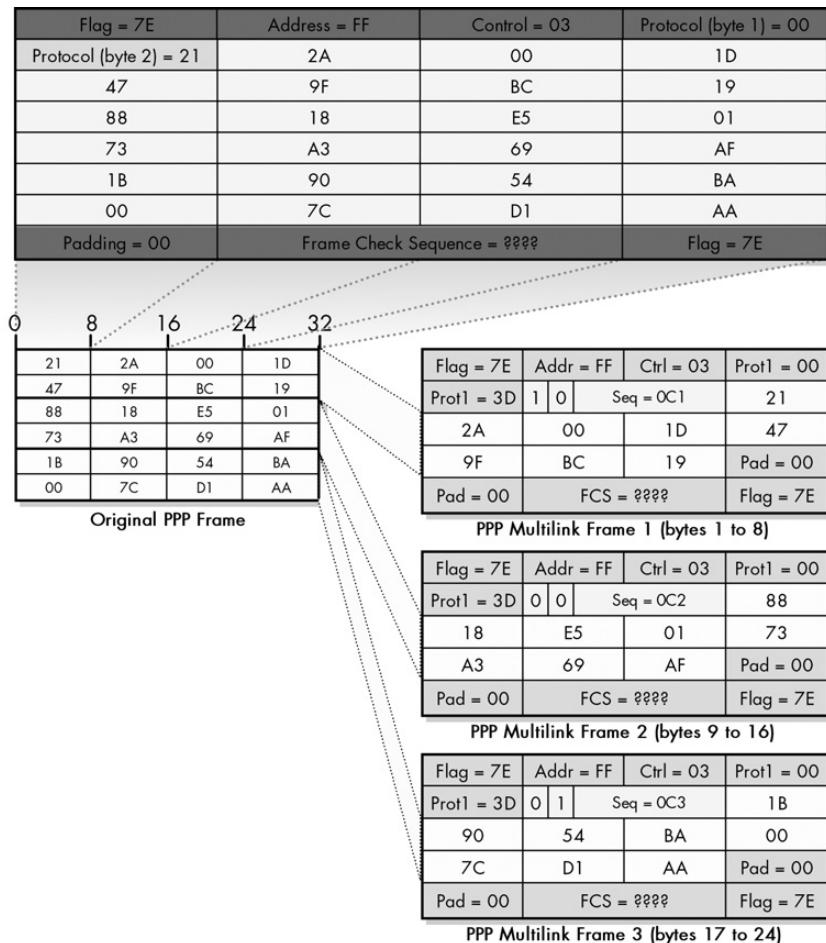
The Fragment Data field contains the actual fragment to be sent. Since the original PPP header (including the Protocol field) is at the start of the original PPP frame, this will appear at the start of the first fragment. The remaining fragments will have just portions of the Information field of the original PPP frame. The last fragment will end with the last bytes of the original PPP frame.

The receiving device will collect all the fragments for each PPP frame and extract the fragment data and MP headers from each. It will use the Sequence Numbers to reassemble the fragments and then process the resulting PPP frame.

### **PPP MP Fragmentation Demonstration**

Figure 12-12 shows a demonstration of fragmenting a PPP data frame. At the top is the same PPP data frame shown in Figure 12-2 earlier in the chapter.

The eight grayed-out bytes are the ones not used when a frame is to be fragmented. Thus, the PPP frame used for MP is 24 bytes long. This frame is split into eight-byte chunks, each of which is carried in the Fragment Data fields of an MP fragment. Note the consecutive Sequence Number values in the fragment frames. Also note that the Beginning Fragment field is set only for the first fragment, and the Ending Fragment is set only for the last one.



**Figure 12-12: PPP MP fragmentation** This diagram shows how a single PPP frame is fragmented into three smaller ones.



# PART II-2

## **TCP/IP NETWORK INTERFACE/ INTERNET LAYER CONNECTION PROTOCOLS**

The second layer of the OSI Reference Model is the *data link layer*; it corresponds to the TCP/IP *network interface layer*. At this layer, most local area network (LAN), wide area network (WAN), and wireless LAN (WLAN) technologies are defined, such as Ethernet and IEEE 802.11.

The third layer of the OSI Reference Model is the *network layer*, also called the *internet layer* in the TCP/IP model. At this layer, internetworking protocols are defined, the most notable being the Internet Protocol (IP).

The second and third layers are intimately related, because messages sent at the network layer must be carried over individual physical networks at the data link layer. They perform different tasks, but as neighbors in the protocol stack, they must cooperate with each other.

A set of protocols serves the important task of linking together these two layers and allowing them to work together. The problem is deciding where exactly these protocols should live. They are sort of the black sheep of the networking world. Nobody denies their importance, but they always think they belong in “the other guy’s” layer. For example, since these protocols pass data on layer 2 networks, the folks who deal with layer 2 technologies say

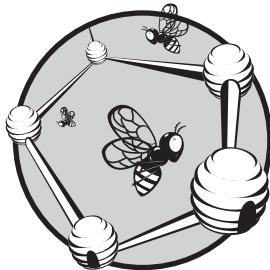
the protocols belong at layer 3. But those who work with layer 3 protocols consider these low-level protocols that provide services to layer 3, and hence put them as part of layer 2.

So where do these protocols go? Well, to some extent, it doesn't really matter. I consider them somewhat special, so I gave them their own home. Welcome to networking layer limbo, also known as OSI model layer 2.5. This is where I put a couple of protocols that serve as glue between the data link and network layers. The main job performed here is address resolution, or providing mappings between layer 2 and layer 3 addresses. This resolution can be done in either direction, and is represented by the two TCP/IP protocols described in this part: the Address Resolution Protocol (ARP) and the Reverse Address Resolution Protocol (RARP), which, despite their similarities, are used for rather different purposes.

I suggest familiarity with the basics of layer 2 and layer 3 (described in Parts I-2 and I-3) before proceeding here. In particular, some understanding of IP addressing is helpful, though not strictly necessary.

# 13

## **ADDRESS RESOLUTION AND THE TCP/IP ADDRESS RESOLUTION PROTOCOL (ARP)**



Communication on an internetwork is accomplished by sending data at layer 3 using a network layer address, but the actual transmission of that data occurs at layer 2 using a data link layer address. This means that every device with a fully specified networking protocol stack will have both a layer 2 and a layer 3 address. It is necessary to define some way of being able to link these addresses together. Usually, this is done by taking a network layer address and determining what data link layer address goes with it. This process is called *address resolution*.

In this chapter, I look at the problem of address resolution at both a conceptual and practical level, with, of course, a focus on how it is done in the TCP/IP protocol suite. I begin with an overview of address resolution in general terms, which describes the issues involved in the process. I then fully describe the TCP/IP Address Resolution Protocol (ARP), probably the

best-known and most commonly used address resolution technique. I then provide a brief look at how address resolution is done for multicast addresses in the Internet Protocol (IP), and finally, the method used in the new IP version 6 (IPv6).

## Address Resolution Concepts and Issues

Due to the prominence of TCP/IP in the world of networking, most discussions of address resolution jump straight to TCP/IP's ARP. This protocol is indeed important, and we will take a look at it later in this chapter. However, the basic problem of address resolution is not unique to any given implementation that deals with it, such as ARP. To provide better understanding of resolving addresses between the data link layer and the network layer and to support our examination of ARP, we'll begin by looking at the matter in more general terms.

I start by discussing the need for address resolution in general terms. I then describe the two main methods for solving the address resolution problem: direct mapping and dynamic resolution. I also explore some of the efficiency issues involved in practical dynamic address resolution, with a focus on the importance of caching.

### The Need for Address Resolution

Some people may balk at the notion of address resolution and the need for protocols that perform this function. In Chapter 5's discussion of the OSI Reference Model, I talked extensively about how the whole point of having conceptual layers was to separate logical functions and allow higher-layer protocols to be hidden from lower-layer details. Given this, why do you need address resolution protocols that tie protocols and layers together?

This is true. However, the OSI Reference Model is exactly that—a *model*. There are often practicalities that arise that require solutions that don't strictly fit the layer model. When the model doesn't fit reality, the model must yield. And so it is in dealing with the problem of address resolution.

### Addressing at Layer 2 and Layer 3

When you consider the seven layers of the OSI Reference Model, there are two that deal with addressing: the data link layer and the network layer. The physical layer is not strictly concerned with addressing at all, but rather, only with sending at the bit level. The layers above the network layer all work with network layer addresses.

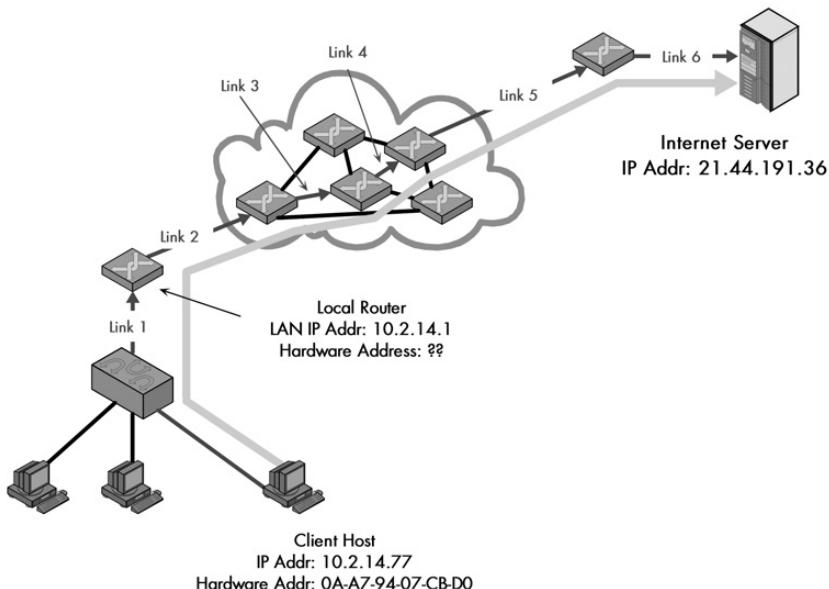
But why is addressing done at two different layers? The answer is that they are very different types of addresses that are used for different purposes. The layer 2 addresses (such as IEEE 802 MAC addresses) are used for local transmissions between hardware devices that can communicate directly. They are used to implement basic local area network (LAN), wireless LAN (WLAN), and wide area network (WAN) technologies. In contrast, layer 3 addresses (most commonly, IP addresses) are used in internetworking to create the equivalent of a massive virtual network at the network layer.

The most important distinction between these types of addresses is between layers 2 and 3: Layer 2 deals with directly connected devices (on the same network), while layer 3 deals with *indirectly* connected devices (as well as directly connected

ones). Say, for example, you want to connect to the web server at <http://www.tcpipguide.com>. This is a website that runs on a server that has an Ethernet card in it that's used for connecting it to its Internet service provider site. However, even if you know its MAC address, you cannot use it to talk directly to this server using the Ethernet card in your home PC, because the devices are on different networks—in fact, they may be on different continents!

Instead, you communicate at layer 3, using the IP and higher-layer protocols such as the Transmission Control Protocol (TCP) and Hypertext Transfer Protocol (HTTP). Your request is *routed* from your home machine, through a sequence of routers to the server at *The TCP/IP Guide*, and the response is routed back to you. The communication is, logically, at layers 3 and above; you send the request not to the MAC address of the server's network card, but rather to the server's IP address.

However, though you can *virtually* connect devices at layer 3, these connections are really conceptual only. When you send a request using IP, it is sent one *hop* at a time, from one physical network to the next. At each of these hops, an actual transmission occurs at the physical and data link layers. When your request is sent to your local router at layer 3, the actual request is encapsulated in a frame using whatever method you physically connect to the router, and then passed to the router using its data link layer address. The same happens for each subsequent step, until finally, the router nearest the destination sends to the destination using its data link (MAC) address. This is illustrated in Figure 13-1.



**Figure 13-1: Why address resolution is necessary** Even though conceptually the client and server are directly connected at layer 3, in reality, information passing between them goes over multiple layer 2 links. In this example, a client on the local network is accessing a server somewhere on the Internet. Logically, this connection can be made directly between the client and server, but in reality, it is a sequence of physical links at layer 2. In this case, there are six such links, most of them between routers that lie between the client and server. At each step, the decision of where to send the data is made based on a layer 3 address, but the actual transmission must be performed using the layer 2 address of the next intended recipient in the route.

The basic problem is that IP addresses are at *too high of a level* for the physical hardware on networks to deal with; they don't understand what they are. When your request shows up at the router that connects to *The TCP/IP Guide*, it can see the <http://www.tcpipguide.com> server's IP address, but that isn't helpful: It needs to send to server's *MAC address*.

The identical issue exists even with communication between devices on a LAN. Even if the web server is sitting on the same desk as the client, the communication is logically at the IP layer, but must also be accomplished at the data link layer. This means you need a way of translating between the addresses at these two layers. This process is called *address resolution*.

**KEY CONCEPT** Address resolution is required because internetworked devices communicate logically using layer 3 addresses, but the actual transmissions between devices take place using layer 2 (hardware) addresses.

### General Address Resolution Methods

In fact, not only do you need to have a way of making this translation, but you need to be concerned with the manner in which it is done. Since the translation occurs for each hop of every datagram sent over an internetwork, the efficiency of the process is extremely important. You don't want to use a resolution method that takes a lot of network resources.

Address resolution can be accomplished in two basic ways: direct mapping and dynamic resolution.

**NOTE** *By necessity, it is not possible to have a fully general address resolution method that works automatically. Since it deals with linking data link layer addresses to network layer addresses, the implementation must be specific to the technologies used in each of these layers. The only method that could really be considered generic would be the use of static, manually updated tables that say, "link this layer 3 address to this layer 2 address." This, of course, is not automatic and brings with it all the limitations of manual configuration.*

### Address Resolution Through Direct Mapping

Network layer addresses must be resolved into data link layer addresses numerous times during the travel of each datagram across an internetwork. You therefore want the process to be as simple and efficient as possible. The easiest method of accomplishing this is to do *direct mapping* between the two types of addresses.

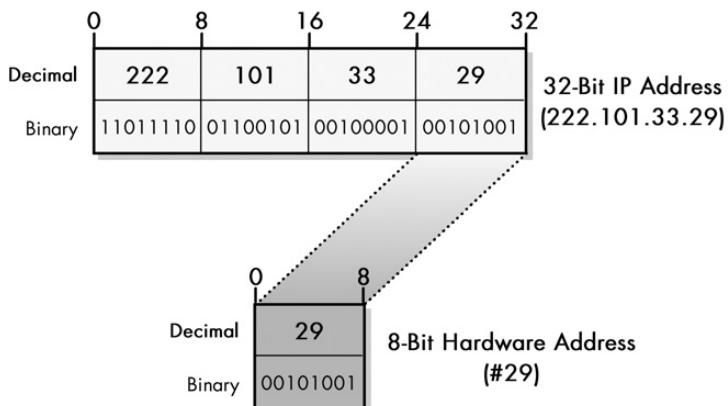
#### How Direct Mapping Works

In the direct mapping technique, a formula is used to map the higher-layer address into the lower-layer address. This is the simpler and more efficient technique, but it has some limitations, especially regarding the size of the data link layer address compared to the network layer address.

The basic idea behind direct mapping is to choose a scheme for layer 2 and layer 3 addresses so that you can determine one from the other using a simple algorithm. This enables you to take the layer 3 address and follow a short procedure to convert it into a layer 2 address. In essence, whenever you have the layer 3 address, you already have the layer 2 address.

The simplest example of direct mapping would be if you used the same structure and semantics for both data link and network layer addresses. This is generally impractical, because the two types of addresses serve different purposes, and are therefore based on incompatible standards. However, you can still perform direct mapping if you have the flexibility of creating layer 3 addresses that are large enough to encode a complete data link layer address within them. Then determining the layer 2 address is a simply matter of selecting a certain portion of the layer 3 address.

As an example, consider a simple LAN technology like ARCNet. It uses a short, 8-bit data link layer address, with valid values of 1 to 255, which can be assigned by an administrator. You could easily set up an IP network on such a LAN by taking a Class C network and using the ARCNet data link layer as the last octet. So, if the network was, for example, 222.101.33.0, you could assign the IP address 222.101.33.1 to the device with ARCNet address #1, the IP address 222.101.33.29 to the device with ARCNet address #29, and so forth, as shown in Figure 13-2.



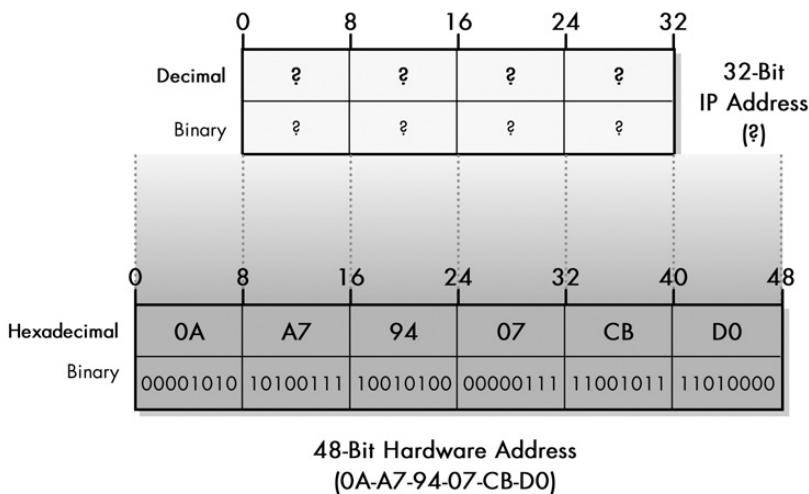
**Figure 13-2: Address resolution through direct mapping** With a small hardware address size, you can easily map each hardware address to a layer 3 address. As you can see in this figure, when the hardware address is small, it is easy to define a mapping that directly corresponds to a portion of a layer 3 address. In this example, an 8-bit MAC address, such as the one used for ARCNet, is mapped to the last byte of the device's IP address, thereby making address resolution a trivial matter.

The appeal of this system is obvious. Conceptually, it is trivial to understand—to get the hardware address for a device, you just use the final eight bits of the IP address. It's also very simple to program devices to perform, and highly efficient, requiring no exchange of data on the network at all.

**KEY CONCEPT** When the layer 2 address is smaller than the layer 3 address, it is possible to define a direct mapping between them so that the hardware address can be determined directly from the network layer address. This makes address resolution extremely simple, but reduces flexibility in how addresses are assigned.

## Problems with Direct Mapping

Unfortunately, direct mapping works only when it is possible to express the data link layer address as a function of the network layer address. Consider instead the same IP address, 222.101.33.29, which is running on an Ethernet network. Here, the data link layer addresses are hardwired into the hardware itself (they can sometimes be overridden, but usually this is not done). More important, the MAC address is 48 bits wide, not 8. This means the layer 2 address is bigger than the layer 3 address, and there is no way to do direct mapping, as Figure 13-3 illustrates. As you can see, when the layer 2 address is larger in size than the layer 3 address, it is not possible to define a mapping between them that can be used for address resolution.



**Figure 13-3: Address resolution problems with large hardware address size** Direct mapping is impossible when the layer 2 address is larger in size than the layer 3 address.

**NOTE** When the hardware address size exceeds the network layer address size, you could do a partial mapping. For example, you could use the IP address to get part of the MAC address and hope you don't have any duplication in the bits you didn't use. This method is not well suited to regular transmissions, but is used for resolving multicast addresses in IPv4 to Ethernet addresses. You'll see how this is done near the end of the chapter.

In general, then, direct mapping is not possible when the layer 3 address is smaller than the layer 2 address. Consider that Ethernet is the most popular technology at layer 2 and uses a 48-bit address, and IP is the most popular technology at layer 3 and uses a 32-bit address. This is one reason why direct mapping is a technique that is not widely used.

What about the next generation of IP? IPv6 supports massive 128-bit addresses (see Chapter 25). Furthermore, regular (unicast) addresses are even defined using a method that creates them from data link layer addresses using a special mapping. This would, in theory, allow IPv6 to use direct mapping for address resolution.

However, the decision was made to have IPv6 use dynamic resolution just as IPv4 does. One reason might be historical, since IPv4 uses dynamic resolution. However, the bigger reason is probably due to a disadvantage of direct mapping: its

inflexibility. Dynamic resolution is a more generalized solution, because it allows data link layer and network layer addresses to be independent, and its disadvantages can be mostly neutralized through careful implementation, as you will see.

In fact, evidence for this can be seen in the fact that dynamic resolution of IP is defined on ARCNet, the example I just used. You could do direct mapping there, but it restricts you to a certain pattern of IP addressing that reduces flexibility.

## **Dynamic Address Resolution**

You just saw that direct mapping provides a simple and highly efficient means of resolving network layer addresses into data link layer addresses. Unfortunately, it is a technique that you either cannot or should not use in a majority of cases. You cannot use it when the size of the data link layer address is larger than that of the network layer address. You shouldn't use it when you need flexibility, because direct mapping requires you to make layer 3 and layer 2 addresses correspond.

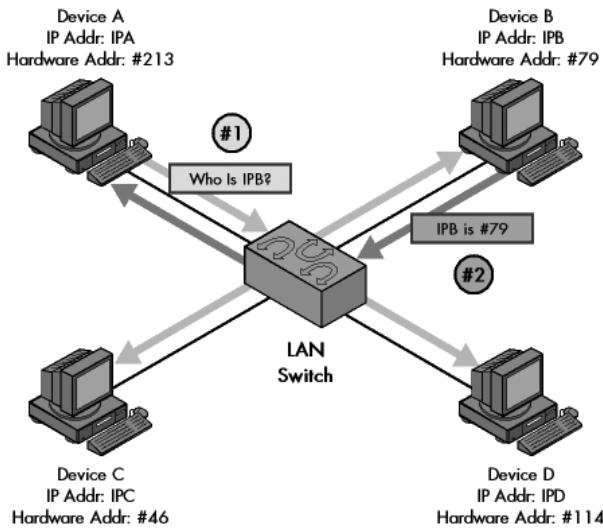
The alternative to direct mapping is a technique called *dynamic address resolution*. This uses a special protocol that allows a device with only an IP address to determine the corresponding data link layer address, even if the two address types take completely different forms. This is normally done by interrogating one or more other devices on a local network to determine what data link layer address corresponds to a given IP address. This is more complex and less efficient than direct mapping, but it's more flexible.

### **How Dynamic Addressing Works**

To understand how dynamic addressing works, you can consider a simple analogy. I'm sure you've seen a limousine driver who is waiting to pick up a person at the airport. (Well, you've seen it in a movie, haven't you?) This is similar to the problem here: The driver knows the name of the person who will be transported, but not the person's face (a type of "local address" in a manner of speaking!). To find the person, the driver holds up a card bearing that person's name. Everyone other than that person ignores the card, but the named individual should recognize it and approach the driver.

You do the same thing with dynamic address resolution in a network. Let's say that Device A wants to send to Device B but knows only Device B's network layer address (its "name") and not its data link layer address (its "face"). It broadcasts a layer 2 frame containing the layer 3 address of Device B—this is like holding up the card with someone's name on it. The devices other than Device B don't recognize this layer 3 address and ignore it. Device B, however, knows its own network layer address. It recognizes this in the broadcast frame and sends a direct response back to Device A. This tells Device A what Device B's layer 2 address is, and the resolution is complete. Figure 13-4 illustrates the process.

**KEY CONCEPT** *Dynamic address resolution* is usually implemented using a special protocol. A device that knows only the network layer address of another device can use this protocol to request the other device's hardware address.



**Figure 13-4: Dynamic address resolution** The device that wants to send data broadcasts a request asking for a response with a hardware address from the other device. Device A needs to send data to Device B, but knows only its IP address (IPB) and not its hardware address. Device A broadcasts a request asking to be sent the hardware address of the device using the IP address IPB. Device B responds back to Device A directly with the hardware address.

Direct mapping is very simple, but as you can see, dynamic resolution isn't exactly rocket science either! It's a simple technique that is easily implemented. Furthermore, it removes the restrictions associated with direct mapping. There is no need for any specific relationship between the network layer address and the data link layer address; they can have a completely different structure and size.

There is one nagging issue though: the efficiency problem. Where direct mapping involves a quick calculation, dynamic resolution requires you to use a protocol to send a message over the network. Fortunately, there are techniques that you can employ to remove some of the sting of this cost through careful implementation.

### Dynamic Address Resolution Caching and Efficiency Issues

You've now seen how dynamic address resolution removes the restrictions that you saw in direct mapping, thereby allowing you to easily associate layer 2 and layer 3 addresses of any size or structure. The only problem with it is that each address resolution requires you to send an extra message that would not be required in direct mapping. Worse yet, since you don't know the layer 2 identity of the recipient, you must use a broadcast message (or at least a multicast), which means that many devices on the local network must take resources to examine the data frame and check which IP address is being resolved.

Sure, sending one extra message may not seem like that big of a deal, and the frame doesn't have to be very large since it contains only a network layer address and some control information. However, when you have to do this for *every* hop of *every* datagram transmission, the overhead really adds up. For this reason, while

basic dynamic address resolution is simple and functional, it's usually not enough. You must add some *intelligence* to the implementation of address resolution in order to reduce the impact on the performance of continual address resolutions.

Consider that most devices on a local network send to only a small handful of other physical devices and tend to do so over and over again. This is a phenomenon known as *locality of reference*, which is observed in a variety of different areas in the computing field. If you send a request to an Internet website from your office PC, it will need to go first to your company network's local router, so you will need to resolve the router's layer 2 address. If you later click a link on that site, that request will also need to go to the router. In fact, almost everything you do off your local network probably goes first to that same router (commonly called a *default gateway*). Having to do a fresh resolution each time would be, well, stupid. It would be like having to look up the phone number of your best friend every time you want to call to say hello.

To avoid being accused of making address resolution protocols that are, well, stupid, designers always include a *caching* mechanism. After a device's network layer address is resolved to a data link layer address, the link between the two is kept in the memory of the device for a period of time. When it needs the layer 2 address the next time, the device just does a quick lookup in its cache. This means that instead of doing a broadcast on every datagram, you do it only once for a whole sequence of datagrams.

Caching is by far the most important performance-enhancing tool in dynamic resolution. It transforms what would otherwise be a very wasteful process into one that, most of the time, is no less efficient than direct mapping. It does, however, add complexity. The cache table entries must be maintained. There is also the problem that the information in the table may become *stale* over time. What happens if you change the network layer address or the data link layer address of a device? For this reason, cache entries must be set to expire periodically. The discussion of caching in TCP/IP's ARP later in this chapter shows some of the particulars of how these issues are handled.

### **Other Enhancements to Dynamic Resolution**

Other enhancements are also possible to the basic dynamic resolution scheme. Let's consider again our example of sending a request to the Internet. You send a request that needs to go to the local router, so you resolve its address and send it the request. A moment later, the reply comes back to the router to be sent to you, so the router needs *your* address. Thus, it would have to do a dynamic resolution on you, even though you just exchanged frames. Again, this is stupid. Instead, you can improve efficiency through *cross-resolution*; when Device A resolves the address of Device B, Device B also adds the entry for Device A to *its* cache.

Another improvement can be made, too. If you think about it, the devices on a local network are going to talk to each other fairly often, even if they aren't chatting right now. If Device A is resolving Device B's network layer address, it will broadcast a frame that Devices C, D, E, and so on all see. Why not have them also update *their* cache tables with resolution information that they see, for future use?

These and other enhancements all serve to cut down on the efficiency problems with dynamic address resolution. They combine to make dynamic resolution close enough to direct mapping in overall capability that there is no good reason not to use it. Once again, you can see some more particulars of this in the section that describes ARP's caching feature.

Incidentally, one other performance-improving idea sometimes comes up during this discussion: Instead of preceding a datagram transmission with an extra broadcast step for address resolution, why not just broadcast the datagram and be done with it? You actually could do this, and if the datagram were small enough, it would be more efficient. Usually, though, datagrams are large, while resolution frames can be quite compact; it makes sense to do a small broadcast and then a large unicast rather than a large broadcast. Also, suppose you did broadcast this one datagram. What about the next datagram and the one after that? Each of these would then need to be broadcast also. When you do a resolution with caching, you need to broadcast only once in a while, instead of continually.

## TCP/IP Address Resolution Protocol (ARP)

ARP is a full-featured, dynamic resolution protocol used to match IP addresses to underlying data link layer addresses. Originally developed for Ethernet, it has now been generalized to allow IP to operate over a wide variety of layer 2 technologies.

**NOTE** *The Address Resolution Protocol described here is used for resolving unicast addresses in version 4 of the Internet Protocol (IPv4). Multicast addresses under IPv4 use a direct mapping method, and IPv6 uses the new Neighbor Discovery (ND) Protocol instead of ARP. These methods are both discussed near the end of this chapter.*

**RELATED INFORMATION** For a discussion of ARP-related issues in networks with mobile IP devices, see Chapter 30.

**RELATED INFORMATION** The software application arp, which is used to administer the TCP/IP ARP implementation on a host, is covered in Chapter 88.

Physical networks function at layers 1 and 2 of the OSI Reference Model and use data link layer addresses. In contrast, internetworking protocols function at layer 3, interconnecting these physical networks to create a possibly huge internetwork of devices specified using network layer addresses. Address resolution is the process whereby network layer addresses are resolved into data link layer addresses. This permits data to be sent one hop at a time across an internetwork.

The problem of address resolution was apparent from the very start in the development of the TCP/IP protocol suite. Much of the early development of IP was performed on the then-fledgling Ethernet LAN technology; this was even before Ethernet had been officially standardized as IEEE 802.3. It was necessary to define a way to map IP addresses to Ethernet addresses to allow communication over Ethernet networks.

As we have already seen in this chapter, there are two basic methods to correlate IP and Ethernet addresses: direct mapping or dynamic resolution. However, Ethernet addresses are 48 bits long, while IP addresses are only 32 bits, which immediately rules out direct mapping. Furthermore, the designers of IP

wanted the flexibility that results from using the dynamic resolution model. To this end, they developed the TCP/IP *Address Resolution Protocol (ARP)*. This protocol is described in one of the earliest of the Internet RFCs still in common use: RFC 826, “An Ethernet Address Resolution Protocol,” which was published in 1982.

The name makes clear that ARP was originally developed for Ethernet. Thus, it represents a nexus between the most popular layer 2 LAN protocol and the most popular layer 3 internetworking protocol. This is true even two decades later. However, it was also obvious from the beginning that even though Ethernet was a very common way of transporting IP, it would not be the only one. Therefore, ARP was made a general protocol that was capable of resolving addresses from IP to Ethernet as well as numerous other data link layer technologies.

The basic operation of ARP involves encoding the IP address of the intended recipient in a broadcast message. It is sent on a local network to allow the intended recipient of an IP datagram to respond to the source with its data link layer address. This is done using a simple request and reply method. A special format is used for ARP messages, which are passed down to the local data link layer for transmission.

**KEY CONCEPT** ARP was developed to facilitate dynamic address resolution between IP and Ethernet and can now be used on other layer 2 technologies as well. It works by allowing an IP device to send a broadcast on the local network, and it requests a response with a hardware address from another device on the same local network.

This basic operation is supplemented by methods to improve performance. Since it was known from the start that having to perform a resolution using broadcast for each datagram was ridiculously inefficient, ARP has always used a cache, where it keeps bindings between IP addresses and data link layer addresses on the local network. Over time, various techniques have been developed to improve the methods used for maintaining cache entries. Refinements and additional features, such as support for cross-resolution by pairs of devices as well as proxy ARP, have also been defined over the years and added to the basic ARP feature set.

### **ARP Address Specification and General Operation**

An ARP transaction begins when a source device on an IP network has an IP datagram to send. It must first decide whether the destination device is on the local network or a distant network. If it's the former, it will send directly to the destination; if it's the latter, it will send the datagram to one of the routers on the physical network for forwarding. Either way, it will determine the IP address of the device that needs to be the immediate destination of its IP datagram on the local network. After packaging the datagram it will pass it to its ARP software for address resolution.

The basic operation of ARP is a *request and response* pair of transmissions on the local network. The source (the one that needs to send the IP datagram) transmits a broadcast containing information about the destination (the intended recipient of the datagram). The destination then responds via unicast back to the source, telling the source the hardware address of the destination.

## ARP Message Types and Address Designations

The terms *source* and *destination* apply to the same devices throughout the transaction. However, there are two different messages sent in ARP: one from the source to the destination and one from the destination to the source. For each ARP message, the *sender* is the one that is transmitting the message and the *target* is the one receiving it. Thus, the identity of the sender and target changes for each message. Here's how the sender and target identities work for requests and replies:

**Request** For the initial request, the sender is the source (the device with the IP datagram to send), and the target is the destination.

**Reply** For the reply to the ARP request, the sender is the destination. It replies to the source, which becomes the target.

Each of the two parties in any message has two addresses (layer 2 and layer 3) to be concerned with, so the following four different addresses are involved in each message:

**Sender Hardware Address** The layer 2 address of the sender of the ARP message.

**Sender Protocol Address** The layer 3 (IP) address of the sender of the ARP message.

**Target Hardware Address** The layer 2 address of the target of the ARP message.

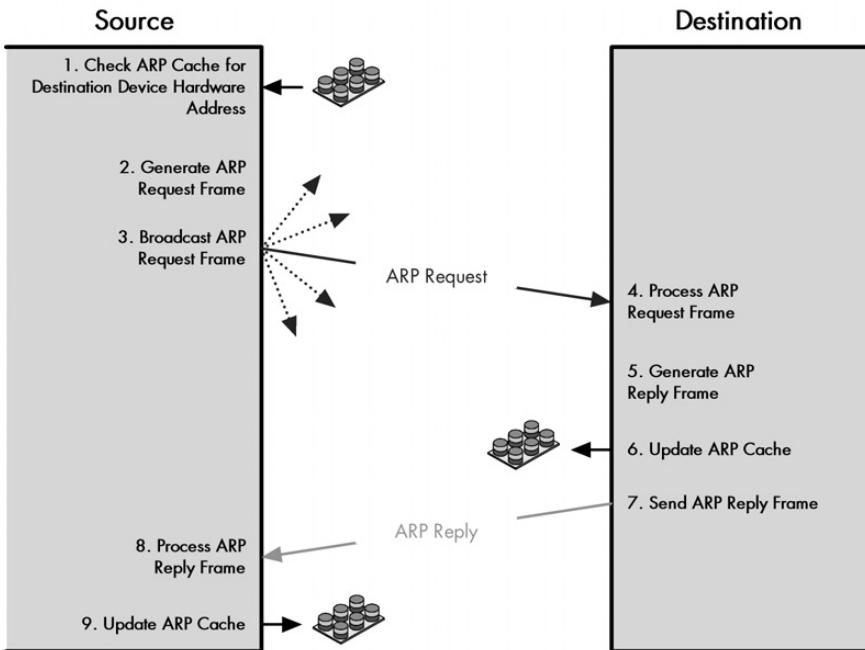
**Target Protocol Address** The layer 3 (IP) address of the target.

These addresses each have a position in the ARP message format, which we'll examine shortly.

## ARP General Operation

With that background in place, let's look at the steps that occur in an ARP transaction. (These steps are also shown graphically in the illustration in Figure 13-5.) This diagram shows the sequence of steps that occur in a typical ARP transaction, as well as the message exchanges between a source and destination device, and the cache checking and update functions. (Incidentally, those little stacks are hard disks, not cans of soup!)

1. **Source Device Checks Cache** The source device will first check its cache to determine if it already has a resolution of the destination device. If so, it can skip to step 9.
2. **Source Device Generates ARP Request Message** The source device generates an ARP Request message. It puts its own data link layer address as the Sender Hardware Address and its own IP address as the Sender Protocol Address. It fills in the IP address of the destination as the Target Protocol Address. (It must leave the Target Hardware Address blank, since that is what it is trying to determine!)
3. **Source Device Broadcasts ARP Request Message** The source broadcasts the ARP Request message on the local network.



**Figure 13-5: Address Resolution Protocol (ARP) transaction process** ARP works by having the source device broadcast a request to find the destination, which responds using a reply message. ARP caches are also consulted and updated as needed.

4. **Local Devices Process ARP Request Message** The message is received by each device on the local network. It is processed, with each device looking for a match on the Target Protocol Address. Those that do not match will drop the message and take no further action.
5. **Destination Device Generates ARP Reply Message** The one device whose IP address matches the contents of the Target Protocol Address of the message will generate an ARP Reply message. It takes the Sender Hardware Address and Sender Protocol Address fields from the ARP Request message and uses these as the values for the Target Hardware Address and Target Protocol Address of the reply. It then fills in its own layer 2 address as the Sender Hardware Address and its IP address as the Sender Protocol Address. Other fields are filled in, as explained in the description of the ARP message format in the following section.
6. **Destination Device Updates ARP Cache** If the source needs to send an IP datagram to the destination now, it makes sense that the destination will probably need to send a response to the source at some point soon. (After all, most communication on a network is bidirectional.) Next, as an optimization, the destination device will add an entry to its own ARP cache that contains the hardware and IP addresses of the source that sent the ARP Request. This saves the destination from needing to do an unnecessary resolution cycle later on.

7. **Destination Device Sends ARP Reply Message** The destination device sends the ARP Reply message. This reply is, however, sent unicast to the source device, because there is no need to broadcast it.
8. **Source Device Processes ARP Reply Message** The source device processes the reply from the destination. It stores the Sender Hardware Address as the layer 2 address of the destination and uses that address for sending its IP datagram.
9. **Source Device Updates ARP Cache** The source device uses the Sender Protocol Address and Sender Hardware Address to update its ARP cache for use in the future when transmitting to this device.

**KEY CONCEPT** ARP is a relatively simple request-and-reply protocol. The source device broadcasts an ARP Request that's looking for a particular device based on the device's IP address. That device responds with its hardware address in an ARP Reply message.

Note that this description goes a bit beyond the basic steps in address resolution, because two enhancements are mentioned. One is caching, which you'll explore shortly. The other is cross-resolution (described earlier in this chapter in the overview of caching issues in dynamic resolution), which is step 6 of the process. This is why the source device includes its IP address in the request. It isn't really needed for any other reason, so you can see that this feature was built into ARP from the start.

### **ARP Message Format**

You've just seen how address resolution is accomplished in ARP, through an exchange of messages between the source device seeking to perform the resolution and the destination device that responds to it. As with other protocols, a special *message format* is used for containing the information required for each step of the resolution process.

ARP messages use a relatively simple format. It includes a field describing the type of message (its *operational code* or *opcode*) and information on both layer 2 and layer 3 addresses. In order to support addresses that may be of varying length, the format specifies the type of protocol used at both layer 2 and layer 3, as well as the length of the addresses used at each of these layers. It then includes space for all four of the address combinations described earlier in this chapter: Sender Hardware Address, Sender Protocol Address, Target Hardware Address, and Target Protocol Address.

The format used for ARP messages is described in Table 13-1. Figure 13-6 shows how the ARP message format is designed to accommodate layer 2 and layer 3 addresses of various sizes. This diagram shows the most common implementation, which uses 32 bits for the layer 3 ("Protocol") addresses and 48 bits for the layer 2 hardware addresses. These numbers correspond to the address sizes of the IPv4 and IEEE 802 MAC addresses that are used by Ethernet.

**Table 13-1:** ARP Message Format

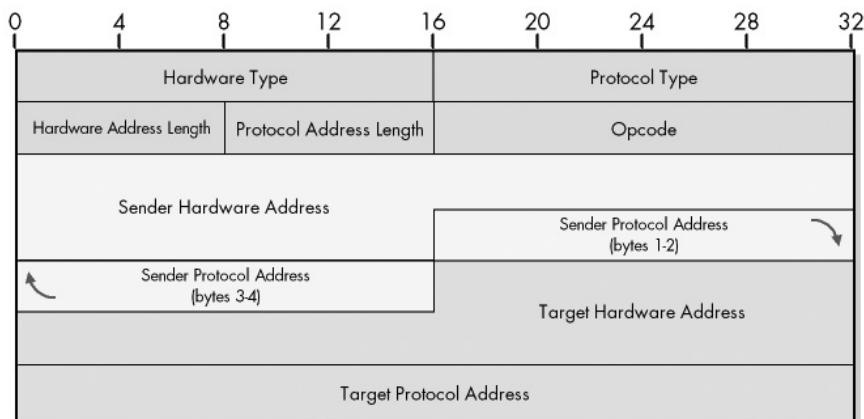
Field Name	Size (Bytes)	Description
<b>HRD</b>	2	Hardware Type: This field specifies the type of hardware used for the local network transmitting the ARP message; thus, it also identifies the type of addressing used. Some of the most common values for this field are shown in Table 13-2.
<b>PRO</b>	2	Protocol Type: This field is the complement of the Hardware Type field, specifying the type of layer 3 addresses used in the message. For IPv4 addresses, this value is 2048 (0800 hex), which corresponds to the EtherType code for IP.
<b>HLN</b>	1	Hardware Address Length: Specifies how long hardware addresses are in this message. For Ethernet or other networks using IEEE 802 MAC addresses, the value is 6.
<b>PLN</b>	1	Protocol Address Length: Again, the complement of the preceding field; specifies how long protocol (layer 3) addresses are in this message. For IPv4 addresses, this value is 4.
<b>OP</b>	2	Opcode: This field specifies the nature of the ARP message being sent. The first two values (1 and 2) are used for regular ARP. Numerous other values are also defined to support other protocols that use the ARP frame format, such as RARP, as shown in Table 13-3. Some protocols are more widely used than others.
<b>SHA</b>	Variable, equals value in HLN field	Sender Hardware Address: The hardware (layer 2) address of the device sending this message, which is the IP datagram source device on a request, and the IP datagram destination on a reply.
<b>SPA</b>	Variable, equals value in PLN field	Sender Protocol Address: The IP address of the device sending this message.
<b>THA</b>	Variable, equals value in HLN field	Target Hardware Address: The hardware (layer 2) address of the device this message is being sent to. This is the IP datagram destination device on a request, and the IP datagram source on a reply.
<b>TPA</b>	Variable, equals value in PLN field	Target Protocol Address: The IP address of the device this message is being sent to.

**Table 13-2:** ARP Hardware Type (HRD) Field Values

Hardware Type (HRD) Value	Hardware Type
1	Ethernet (10 Mb)
6	IEEE 802 Networks
7	ARCNet
15	Frame Relay
16	Asynchronous Transfer Mode (ATM)
17	HDLC
18	Fibre Channel
19	Asynchronous Transfer Mode (ATM)
20	Serial Line

**Table 13-3: ARP Opcode (OP) Field Values**

Opcode	ARP Message Type
1	ARP Request
2	ARP Reply
3	RARP Request
4	RARP Reply
5	DRARP Request
6	DRARP Reply
7	DRARP Error
8	InARP Request
9	InARP Reply



**Figure 13-6: ARP message format**

Once the ARP message has been composed, it is passed down to the data link layer for transmission. The entire contents of the ARP message become the payload for the message actually sent on the network, such as an Ethernet frame on an Ethernet LAN. Note that the total size of the ARP message is variable, since the address fields are of variable length. Normally, though, these messages are quite small. For example, they are only 28 bytes for a network carrying IPv4 datagrams in IEEE 802 MAC addresses.

### **ARP Caching**

ARP is a dynamic resolution protocol, which means that every resolution requires the interchange of messages on the network. Each time a device sends an ARP message, it ties up the local network, consuming network bandwidth that cannot be used for other traffic. ARP messages aren't large, but having to send them for every hop of every IP datagram would represent an unacceptable performance hit on the network. It also wastes time compared to the simpler direct mapping method of

resolution. On top of this, the ARP Request message is broadcasted, which means every device on the local network must spend CPU time examining the contents of each one.

The general solution to the efficiency issues with dynamic resolution is to employ *caching*. In addition to reducing network traffic, caching also ensures that the resolution of commonly used addresses is fast, thereby making overall performance comparable to direct mapping. For this reason, caching functionality has been built into ARP from the start.

### **Static and Dynamic ARP Cache Entries**

The ARP cache takes the form of a table containing matched sets of hardware and IP addresses. Each device on the network manages its own ARP cache table. There are two different ways that cache entries can be put into the ARP cache:

**Static ARP Cache Entries** These are address resolutions that are manually added to the cache table for a device and are kept in the cache on a permanent basis. Static entries are typically managed using a tool such as the arp software utility (see Chapter 88).

**Dynamic ARP Cache Entries** These are hardware and IP address pairs that are added to the cache by the software itself as a result of past ARP resolutions that were successfully completed. They are kept in the cache for only a period of time and are then removed.

A device’s ARP cache can contain both static and dynamic entries, each of which has advantages and disadvantages. However, dynamic entries are used most often because they are automatic and don’t require administrator intervention.

Static ARP entries are best used for devices that a given device needs to communicate with on a regular basis. For example, a workstation might have a static ARP entry for its local router and file server. Since the entry is static, it is always found in step 1 of the ARP transaction process, and there is no need to ever send resolution messages for the destination in that entry. The disadvantage is that these entries must be manually added, and they must also be changed if the hardware or IP addresses of any of the hardware in the entries change. Also, each static entry takes space in the ARP cache, so you don’t want to overuse static entries. It wouldn’t be a good idea to have static entries for every device on the network, for example.

### **Cache Entry Expiration**

Dynamic entries are added automatically to the cache on an as-needed basis, so they represent mappings for hosts and routers that a given device is actively using. They do not need to be manually added or maintained. However, it is also important to realize that dynamic entries cannot be added to the cache and left there forever—dynamic entries left in place for a long time can become stale.

Consider Device A’s ARP cache, which contains a dynamic mapping for Device B, which is another host on the network. If dynamic entries stayed in the cache forever, the following situations might arise.

**Device Hardware Changes** Device B might experience a hardware failure that requires its network interface card to be replaced. The mapping in Device A’s cache would become invalid, since the hardware address in the entry is no longer on the network.

**Device IP Address Changes** Similarly, the mapping in Device A’s cache also would become invalid if Device B’s IP address changed.

**Device Removal** Suppose Device B is removed from the local network. Device A would never need to send to it again at the data link layer, but the mapping would remain in Device A’s cache, wasting space and possibly taking up search time.

To avoid these problems, dynamic cache entries must be set to automatically expire after a period of time. This is handled automatically by the ARP implementation, with typical timeout values being 10 or 20 minutes. After a particular entry times out, it is removed from the cache. The next time that address mapping is needed, a fresh resolution is performed to update the cache. This is very slightly less efficient than static entries, but sending two 28-byte messages every 10 or 20 minutes isn’t a big deal.

As mentioned in the overview of ARP operation, dynamic cache entries are added not only when a device initiates a resolution, but when it is the destination device as well. This is another enhancement that reduces unnecessary address resolution traffic.

### Other Caching Features

Other enhancements are also typically put into place, depending on the implementation. Standard ARP requires that if Device A initiates resolution with a broadcast, each device on the network should update its own cache entries for Device A, even if they are not the device that Device A is trying to reach. However, these “third-party” devices are *not* required to create new cache entries for Device A in this situation.

The issue here is a trade-off. Creating a new cache entry would save any of those devices from needing to resolve Device A’s address in the near future. However, it also means every device on the network will quickly have an ARP cache table filled up with the addresses of most of the other devices on the network. This may not be desirable in larger networks. Even in smaller ones, this model may not make sense, given that modern computing is client/server in nature and peer devices on a LAN may not often communicate directly. Some devices may choose to create such cache entries, but they may set them to expire after a very short time to avoid filling the cache.

Each ARP implementation is also responsible for any other housekeeping required to maintain the cache. For example, if a device is on a local network with many hosts and its cache table is too small, it might be necessary for older, less frequently used entries to be removed to make room for newer ones. Ideally, the cache should be large enough to hold all the other devices with which a device communicates on a regular basis on the network, along with some room for ones it occasionally talks to.

## Proxy ARP

ARP was designed to be used by devices that are directly connected on a local network. Each device on the network should be capable of sending both unicast and broadcast transmissions directly to one another. Normally, if Device A and Device B are separated by a router, they would not be considered local to each other. Device A would not send directly to Device B or vice versa; they would send to the router instead at layer 2 and would be considered two hops apart at layer 3.

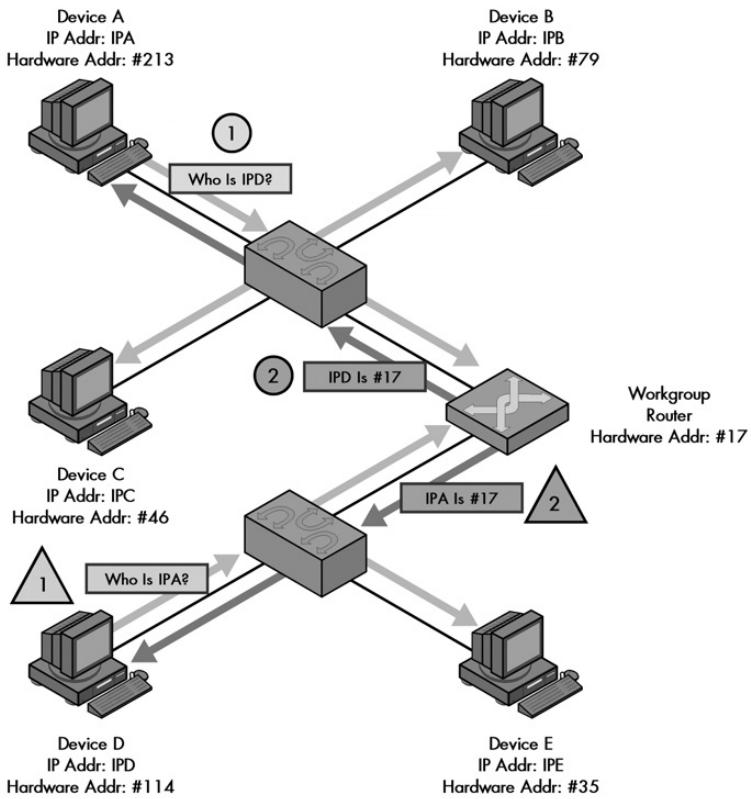
In some networking situations, however, there might be two physical network segments that are in the same IP network or subnet and are connected by a router. In other words, Device A and Device B might be on different networks at the data link layer level, but on the same IP network or subnet. When this happens, Device A and Device B will each think the other is on the local network when they look to send IP datagrams.

In this situation, suppose that Device A wants to send a datagram to Device B. It doesn't have Device B's hardware address in the cache, so it begins an address resolution. When it broadcasts the ARP Request message to get Device B's hardware address, however, it will quickly run into a problem: Device B is not on Device A's local network. The router between them will not pass Device A's broadcast onto Device B's part of the network, because routers don't pass hardware-layer broadcasts. Device B will never get the request, and thus Device A will not get a reply containing Device B's hardware address.

The solution to this situation is called *ARP proxying* or *Proxy ARP*. In this technique, the router that sits between the local networks is configured to respond to Device A's broadcast on behalf of Device B. It does not send back to Device A the hardware address of Device B. Since they are not on the same network, Device A cannot send directly to Device B anyway. Instead, the router sends Device A its own hardware address. Device A then sends to the router, which forwards the message to Device B on the other network. Of course, the router also does the same thing on Device A's behalf for Device B, and for every other device on both networks, when a broadcast is sent that targets a device that isn't on the same actual physical network as the resolution initiator. This is illustrated in Figure 13-7.

Proxy ARP provides flexibility for networks where hosts are not all actually on the same physical network but are configured as if they were at the network layer. It can be used to provide support in other special situations where a device cannot respond directly to ARP message broadcasts. It may be used when a firewall is configured for security purposes. A type of proxying is also used as part of Mobile IP to solve the problem of address resolution when a mobile device travels away from its home network.

**KEY CONCEPT** Since ARP relies on broadcasts for address resolution, and broadcasts are not propagated beyond a physical network, ARP cannot function between devices on different physical networks. When such operation is required, a device, such as a router, can be configured as an ARP proxy to respond to ARP requests on the behalf of a device on a different network.



**Figure 13-7: ARP Proxy operation** These two examples show how a router acting as an ARP proxy returns its own hardware address in response to requests by one device for an address on the other network. In this small internetwork shown, a single router connects two LANs that are on the same IP network or subnet. The router will not pass ARP broadcasts, but has been configured to act as an ARP proxy. In this example, Device A and Device D are each trying to send an IP datagram to the other, and so each broadcasts an ARP Request. The router responds to the request sent by Device A as if it were Device D, giving to Device A its own hardware address (without propagating Device A's broadcast). It will forward the message sent by Device A to Device D on Device D's network. Similarly, it responds to Device D as if it were Device A, giving its own address, then forwarding what Device D sends to it over to the network where Device A is located.

The main advantage of proxying is that it is transparent to the hosts on the different physical network segments. The technique has some drawbacks, however. First, it introduces added complexity. Second, if more than one router connects two physical networks using the same network ID, problems may arise. Third, it introduces potential security risks; since it essentially means that a router impersonates devices by acting as a proxy for them, the potential for a device spoofing another is real. For these reasons, it may be better to redesign the network so that routing is done between physical networks separated by a router, if possible.

## TCP/IP Address Resolution for IP Multicast Addresses

Like most discussions of address resolution, most of this chapter so far has focused on unicast communication, where a datagram is sent from one source device to one destination device. Whether direct mapping or dynamic resolution is used for resolving a network layer address, it is a relatively simple matter to resolve addresses when there is only one intended recipient of the datagram. As you've seen, TCP/IP uses ARP for its dynamic resolution scheme, which is designed for unicast resolution only.

However, IP also supports *multicasting* of datagrams, as I explain in the sections on IP multicasting and IP multicast addressing in Chapters 23 and 17, respectively. In this situation, the datagram must be sent to multiple recipients, which complicates matters considerably. You need to establish a relationship of some sort between the IP multicast group address and the addresses of the devices at the data link layer. You could do this by converting the IP multicast datagram to individual unicast transmissions at the data link layer with each using ARP for resolution, but this would be horribly inefficient.

When possible, IP makes use of the multicast addressing and delivery capabilities of the underlying network to deliver multicast datagrams on a physical network. Perhaps surprisingly, even though ARP employs dynamic resolution, multicast address resolution is done using a version of the direct mapping technique. By defining a *mapping* between IP multicast groups and data link layer multicast groups, you enable physical devices to know when to pay attention to multicasted datagrams.

The most commonly used multicast-capable data link addressing scheme is the IEEE 802 addressing system best known for its use in Ethernet networks. These data link layer addresses have 48 bits, arranged into two blocks of 24. The upper 24 bits are arranged into a block called the *organizationally unique identifier (OUI)*, with different values assigned to individual organizations; the lower 24 bits are then used for specific devices.

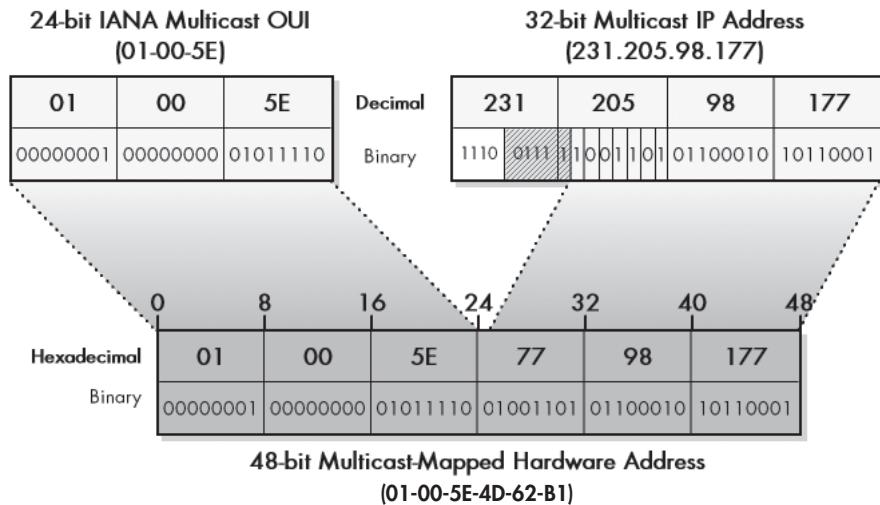
The Internet Assigned Number Authority (IANA) itself has an OUI that it uses for mapping multicast addresses to IEEE 802 addresses. This OUI is 01:00:5E. To form a mapping for Ethernet, 24 bits are used for this OUI, and the 25th (of the 48) is always zero. This leaves 23 bits of the original 48 to encode the multicast address. To do the mapping, the lower-order 23 bits of the multicast address are used as the last 23 bits of the Ethernet address starting with 01:00:5E for sending the multicast message.

Figure 13-8 illustrates how the multicast address mapping process works.

**KEY CONCEPT** IP multicast addresses are resolved to IEEE 802 (Ethernet) MAC addresses using a direct mapping technique that uses 23 of the 28 bits in the IP multicast group address.

Of course, there are 28 unique bits in IP multicast addresses, so this is a "bit" of a problem! What it means is that there is no unique mapping between IP multicast addresses and Ethernet multicast addresses. Since 5 of the 28 bits of the multicast group cannot be encoded in the Ethernet address, 32 ( $2^5$ ) different IP multicast

addresses map onto each possible Ethernet multicast address. In theory, this would be a problem, but in practice, it isn't. The chances of any two IP multicast addresses on a single network mapping to the same Ethernet multicast address at the same time are pretty small.



**Figure 13-8: Mapping of multicast IP addresses to IEEE 802 multicast MAC addresses** Multicast IP addresses are mapped to IEEE 802 multicast MAC addresses by copying the IANA multicast OUI value (01-00-5E) to the top 24 bits, setting the 25th bit to zero, and copying the bottom 23 bits of the multicast address to the remaining 23 bits. To create a 48-bit multicast IEEE 802 (Ethernet) address, the top 24 bits are filled in with the IANA's multicast OUI, 01-00-5E. The 25th bit is zero, and the bottom 23 bits of the multicast group are put into the bottom 23 bits of the MAC address. This leaves 5 bits (shown hatched) that are not mapped to the MAC address, meaning that 32 different IP addresses may have the same mapped multicast MAC address.

Still, it is possible that two IP multicast groups might be in use on the same physical network and might map to the same data link layer multicast address. For this reason, devices must not assume that all multicast messages they receive are for their groups; they must pass up the messages to the IP layer to check the full IP multicast address to make sure that they really were supposed to get the multicast datagram they received. If they accidentally get one that was intended for a multicast group they are not a member of, they discard it. This happens infrequently, so the relative lack of efficiency is not a large concern.

## TCP/IP Address Resolution for IP Version 6

The TCP/IP ARP is a fairly generic protocol for dynamically resolving network layer addresses into data link layer addresses. Even though it was designed for IPv4, the message format allows for variable-length addresses at both the hardware and network layers. This flexibility means it would have been theoretically possible to use it for the new version of IP, IPv6. Some minor changes might have been required, but the technique could have been about the same.

The designers of IPv6 chose not to do this, however. Changing IP is a big job that has been under way for many years, providing a rare opportunity to change various aspects of TCP/IP. The Internet Engineering Task Force (IETF) decided to take advantage of the changes in IPv6 to overhaul not only IP itself, but also many of the protocols that support or assist it. In IPv6, the address resolution job of ARP has been combined with several functions performed by the Internet Control Message Protocol (ICMP) in the original TCP/IP suite, supplemented with additional capabilities and defined as the new Neighbor Discovery (ND) Protocol.

The term *neighbor* in IPv6 simply refers to devices on a local network, and as the name implies, ND is responsible for tasks related to communicating information between neighbors (among other things). I describe ND briefly in Chapter 36, including a discussion of the various tasks it performs. Here, I focus specifically on how ND performs address resolution.

The basic concepts of address resolution in IPv6 ND aren't all that different from those in IPv4 ARP. Resolution is still dynamic and is based on the use of a cache table that maintains pairings of IPv6 addresses and hardware addresses. Each device on a physical network keeps track of this information for its neighbors. When a source device needs to send an IPv6 datagram to a local network neighbor but doesn't have its hardware address, it initiates the resolution process. For clarity in the text let's say that, as usual, Device A is trying to send to Device B.

Instead of sending an ARP Request message, Device A creates an ND Neighbor Solicitation message. Now, here's where the first big change can be seen from ARP. If the underlying data link protocol supports multicasting, as Ethernet does, the Neighbor Solicitation message is not broadcast. Instead, it is sent to the *solicited-node address* of the device whose IPv6 address you are trying to resolve. So Device A won't broadcast the message, but it will multicast it to Device B's solicited-node multicast address.

The solicited-node multicast address is a special mapping that each device on a multicast-capable network creates from its unicast address; it is described in Chapter 25's discussion of IPv6 multicast addresses. The solicited-node address isn't unique for every IPv6 address, but the odds of any two neighbors on a given network having the same one are small. Each device that receives a multicasted Neighbor Solicitation must still check to make sure it is the device whose address the source is trying to resolve.

Why bother with this, if devices still have to check each message? The multicast will affect at most a small number of devices. With a broadcast, each and every device on the local network would receive the message, while the use of the solicited-node address means at most that a couple of devices will need to process it. Other devices don't even have to bother checking the Neighbor Solicitation message at all.

Device B will receive the Neighbor Solicitation and respond back to Device A with a Neighbor Advertisement. This is analogous to the ARP Reply and tells Device A the physical address of Device B. Device A then adds Device B's information to its neighbor cache. For efficiency, cross-resolution is supported, as in IPv4 address resolution. This is done by having Device A include its own layer 2 address in the Neighbor Solicitation, assuming it knows it. Device B will record this along with Device A's IP address in Device B's neighbor cache.

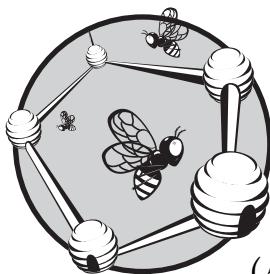
**KEY CONCEPT** Address resolution in IPv6 uses the new *Neighbor Discovery (ND) Protocol* instead of the Address Resolution Protocol (ARP). A device trying to send an IPv6 datagram sends a Neighbor Solicitation message to get the address of another device, which responds with a Neighbor Advertisement. When possible, to improve efficiency, the request is sent using a special type of multicast address rather than broadcast.

This is actually a fairly simplified explanation of how resolution works in IPv6, because ND is quite complicated. Neighbor solicitations and advertisements are also used for other functions, such as testing the reachability of nodes and determining if duplicate addresses are in use. There are many special cases and issues that ND addresses to ensure that no problems develop during address resolution. ND also supports proxied address resolution.

**NOTE** Even though I put this discussion where it would be near the other discussions of address resolution, ND really isn't a layer connection or lower-level protocol like ARP. It is analogous to ICMP (Chapter 31) in its role and function, and, in fact, makes use of ICMP(v6) messages. One advantage of this architectural change is that there is less dependence on the characteristics of the physical network, so resolution is accomplished in a way that's more similar to other network support activities. Thus, it is possible to make use of facilities that can be applied to all IP datagram transmissions, such as IP security features. Chapter 36 contains much more information on this subject.

# 14

## **REVERSE ADDRESS RESOLUTION AND THE TCP/IP REVERSE ADDRESS RESOLUTION PROTOCOL (RARP)**



In Chapter 13, you explored the operation of the TCP/IP Address Resolution Protocol (ARP). ARP is used when a device needs to determine the layer 2 (hardware) address of some other device but has only its layer 3 (network, IP) address. It broadcasts a hardware layer request, and the target device responds with the hardware address that matches the known IP address.

In theory, it is also possible to use ARP in the opposite way. If you know the hardware address of a device but not its IP address, you could broadcast a request containing the hardware address and get back a response that contains the IP address. In this chapter, you will briefly explore this concept of *reverse address resolution*.

The obvious first question is why would you ever need to do this? Since you are dealing with communication on an Internet Protocol (IP) internetwork, you are always going to know the IP address of the destination of the datagram you need to send—it's right there in the datagram itself. You also know your own IP address as well. Or do you?

In a traditional TCP/IP network, every normal host on a network knows its IP address because it is stored somewhere on the machine. When you turn on your PC, the TCP/IP software reads the IP address from a file, which allows your PC to learn and start using its IP address. However, there are some devices, such as diskless workstations, that don't have any means of storing an IP address where it can be easily retrieved. When these units are powered up, they know their physical address only (because it's wired into the hardware) but not their IP address.

The problem you need to solve here is what is commonly called *bootstrapping* in the computer industry. This refers to the concept of starting something from a zero state; it is analogous to "pulling yourself up by your own bootstraps." This is seemingly impossible, just as it seems paradoxical to use TCP/IP to configure the IP address that is needed for TCP/IP communications. However, it is indeed possible to do this, by making use of broadcasts, which allow local communication even when the target's address is not known.

## The Reverse Address Resolution Protocol (RARP)

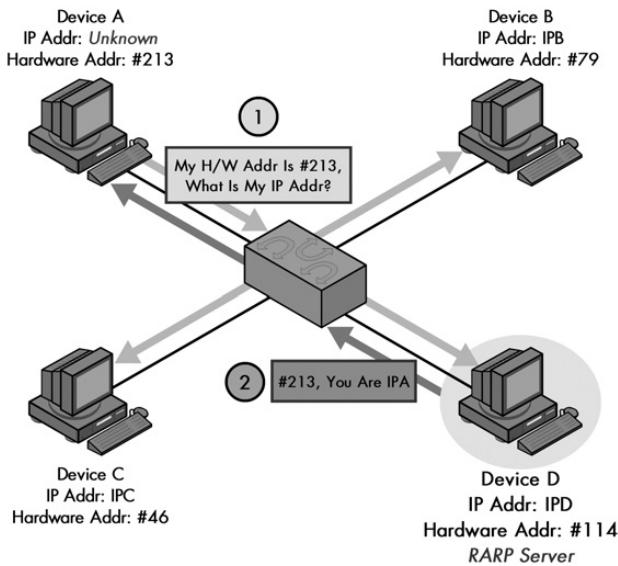
The first method devised to address the bootstrapping problem in TCP/IP was the backward use of ARP, which is described in the previous chapter. This technique was formalized in RFC 903, "A Reverse Address Resolution Protocol (RARP)," published in 1984. ARP allows Device A to say, "I am Device A, and I have Device B's IP address. Device B please tell me your hardware address." RARP is used by Device A to say, "I am Device A, and I am sending this broadcast using my hardware address; can someone please tell me *my* IP address?"

The two-step operation of RARP is illustrated in Figure 14-1. As the name suggests, RARP works like ARP but in reverse, which is why this diagram is similar to Figure 13-4.

The next question then is who knows Device A's IP address if Device A doesn't? The answer is that a special *RARP server* must be configured to listen for RARP requests and then issue replies to them. Each physical network where RARP is in use must have RARP software running on at least one machine.

RARP is not only very similar to ARP, it basically *is* ARP. RFC 903 doesn't define a whole new protocol from scratch; it just describes a new method for using ARP to perform the opposite of its normal function. RARP uses ARP messages in the same format as ARP (described in Chapter 13), but uses different opcodes to accomplish its reverse function. As in ARP, a request and reply are used in an exchange. The meaning of the address fields is the same, too: The sender is the device transmitting a message, while the target is the one receiving it.

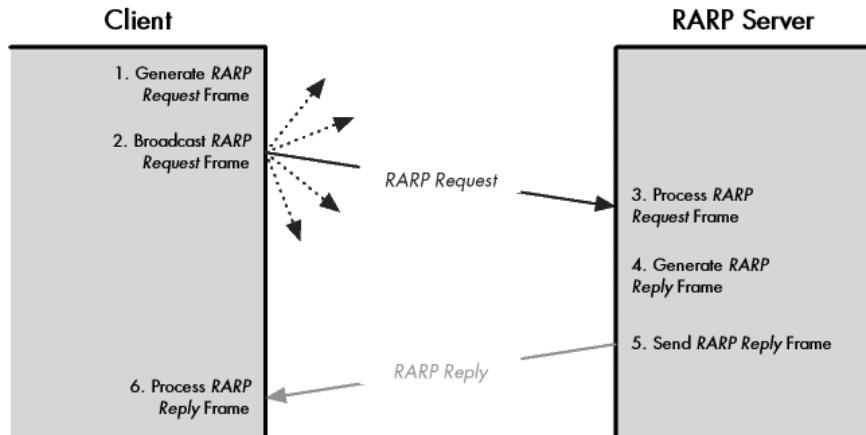
**KEY CONCEPT** The Reverse Address Resolution Protocol (RARP) is the earliest and simplest protocol that's designed to allow a device to obtain an IP address for use on a TCP/IP network. It is based directly on ARP and works in basically the same way, but in reverse: A device sends a request containing its hardware address, and a device set up as an RARP server responds back with the device's assigned IP address.



**Figure 14-1: Operation of the Reverse Address Resolution Protocol (RARP)** RARP works like ARP but in reverse; a device broadcasts its hardware address and an RARP server responds with its IP address. Here, instead of Device A providing the IP address of another device and asking for its hardware address, it is providing its own hardware address and asking for an IP address it can use. The answer, in this case, is provided by Device D, which is serving as an RARP server for this network.

## RARP General Operation

Figure 14-2 shows the steps followed in a RARP transaction. As you can see, RARP uses a simple request and reply exchange to allow a device to obtain an IP address.



**Figure 14-2: Reverse Address Resolution Protocol (RARP) operation** RARP consists of the exchange of one broadcast request message and one unicast reply message.

Here's what happens at each step:

1. **Source Device Generates RARP Request Message** The source device generates an RARP Request message. Thus, it uses the value 3 for the *opcode* in the message. It puts its own data link layer address as both the Sender Hardware Address and also the Target Hardware Address. It leaves both the Sender Protocol Address and the Target Protocol Address blank, since it doesn't know either.
2. **Source Device Broadcasts RARP Request Message** The source broadcasts the ARP Request message on the local network.
3. **Local Devices Process RARP Request Message** The message is received by each device on the local network and processed. Devices that are not configured to act as RARP servers ignore the message.
4. **RARP Server Generates RARP Reply Message** Any device on the network that is set up to act as an RARP server responds to the broadcast from the source device. It generates an RARP Reply using an opcode value of 4. It sets the Sender Hardware Address and Sender Protocol Address to its own hardware and IP address, since it is the sender of the reply. It then sets the Target Hardware Address to the hardware address of the original source device. It looks up in a table the hardware address of the source, determines that device's IP address assignment, and puts it into the Target Protocol Address field.
5. **RARP Server Sends RARP Reply Message** The RARP server sends the RARP Reply message unicast to the device looking to be configured.
6. **Source Device Processes RARP Reply Message** The source device processes the reply from the RARP server. It then configures itself using the IP address in the Target Protocol Address supplied by the RARP server.

**NOTE** *More than one RARP server may respond to a request, if two or more are configured on any local network. The source device will typically use the first reply and discard the others.*

## Limitations of RARP

RARP is the earliest and most rudimentary of the class of technologies I call *host configuration protocols*, which I describe in general terms in Chapter 59. As the first of these protocols, RARP was a useful addition to TCP/IP in the early 1980s, but has several shortcomings, the most important of which are as follows:

**Low-Level Hardware Orientation** RARP works using hardware broadcasts. This means that if you have a large internetwork with many physical networks, you need an RARP server on *every* network segment. Worse, if you need reliability to make sure RARP keeps running even if one RARP server goes down, you need *two* on each physical network. This makes centralized management of IP addresses difficult.

**Manual Assignment** RARP allows hosts to configure themselves automatically, but the RARP server must still be set up with a manual table of bindings between hardware and IP addresses. These must be maintained for each server, which is, again, a lot of work for an administrator.

**Limited Information** RARP provides a host with only its IP address. It cannot provide other needed information such as, for example, a subnet mask or default gateway.

The importance of host configuration has increased dramatically since the early 1980s. Many organizations assign IP addresses dynamically even for hosts that have disk storage, because of the many advantages this provides in administration and because of the efficient use of address space. For this reason, RARP has been replaced by two more capable technologies that operate at higher layers in the TCP/IP protocol stack: BOOTP and DHCP. They are discussed in the application layer section on host configuration protocols, in Chapters 60 through 64.



# PART II-3

## INTERNET PROTOCOL VERSION 4 (IP/IPv4)

The idea of singling out any one protocol as being more important than the others in a network is kind of pointless, if you think about it. The protocols and technologies work as a team to accomplish the goal of communication across the network. As with any team, no single member can get the job done alone, no matter how good it is. Still, if we were to try to pick a “most valuable player” in the world of networking, a good case could be made that we have it in the TCP/IP *Internet Protocol (IP)*.

Even though it gets second billing in the name of the TCP/IP protocol suite, IP is the workhorse of TCP/IP. It implements key network layer functions including addressing, datagram handling, and routing, and it is the foundation on which other TCP/IP protocols are built. Even the ones lower in the TCP/IP architecture, such as the Address Resolution Protocol (ARP) and the Point-to-Point Protocol (PPP), are easier to understand when you know how IP works.

This part includes nine chapters that provide considerable coverage of IP. The first chapter gives an overview of IP as a whole, including a discussion of its versions, while the rest of the chapters focus on the details of operation of the most popular current version of the protocol, *IP version 4 (IPv4)*.

The second through sixth chapters discuss in great detail the concepts and practice behind IP addressing. The second chapter provides an overview of IPv4 addressing concepts and issues. The third discusses the original,

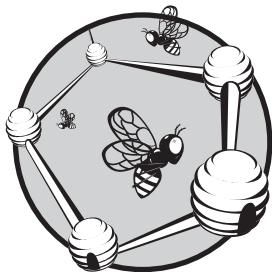
class-based (*classful*) IP addressing scheme and how the different classes work. The fourth and fifth chapters are devoted to IP subnets and subnet addressing. They discuss subnetting concepts and include an illustration of practical step-by-step subnetting. The sixth chapter describes the new classless addressing system, also sometimes called *supernetting*.

The seventh through ninth chapters discuss important practical issues related to how IPv4 datagrams are created and handled. You'll find a full description of the IPv4 message format and options in the seventh chapter; explanations of IP datagram sizing, fragmentation, and reassembly in the eighth chapter; and coverage of routing and multicasting in the ninth chapter.

As the title of this part implies, the coverage here is limited to IPv4. (For simplicity, in this part, I use the simpler designation *IP* rather than *IPv4*, except where the version number is required for clarity.) IP version 6 (IPv6) is covered in its separate section (Part II-4), as are the IP-related protocols. That said, some of the principles here will also apply to IPv6, as well as IP Network Address Translation (NAT), IPsec, and Mobile IP (Part II-4) in a limited manner.

# 15

## **INTERNET PROTOCOL VERSIONS, CONCEPTS, AND OVERVIEW**



The Internet Protocol (IP) is a very important protocol in internetworking. It would be no exaggeration to say that you can't really comprehend modern networking without a good understanding of IP. Unfortunately, IP can be somewhat difficult to understand. A large amount of complexity has become associated with it over the years, and this has allowed it to meet the many demands placed on it.

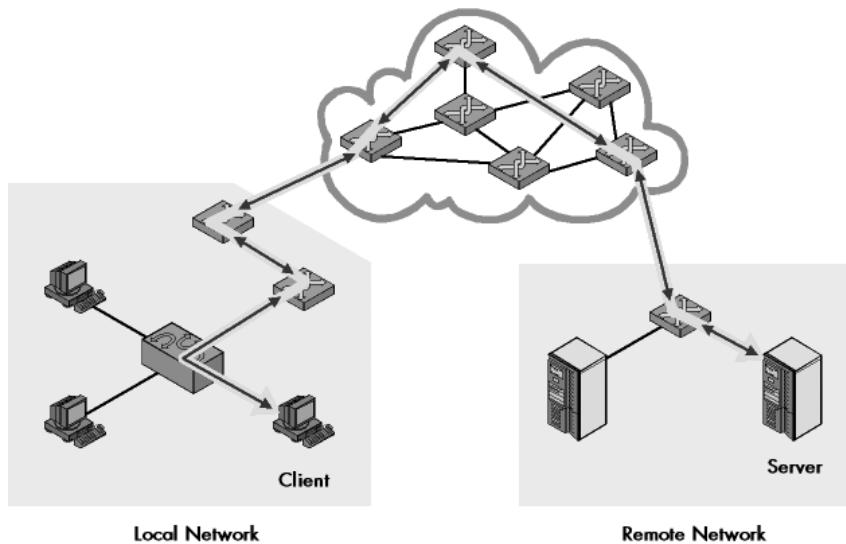
Before diving into the details of how IP works, we'll look at the basic concepts underlying IP. In this chapter, I explain how IP operates in basic terms and the most important aspects of how it does its job. We'll look at its main functions, its history, and how it has spawned the development of several IP-related protocols.

## IP Overview and Key Operational Characteristics

IP is the core of the TCP/IP protocol suite and the main protocol at the network layer. The network layer is primarily concerned with the delivery of data between devices that may be on different networks, which are interconnected in an arbitrary manner. In other words, an *internetwork*. IP is the mechanism by which this data is sent on TCP/IP networks (with help from other protocols at the network layer, too, of course).

Let's look at the TCP/IP layer model and consider what IP does from an architectural standpoint. As the layer 3 protocol, it provides a service to layer 4 in the TCP/IP stack, represented mainly by the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) (see Part II-8). IP takes data that has been packaged by either TCP or UDP, manipulates it as necessary, and sends it out (see Figure 15-1).

This service is sometimes called *internetwork datagram delivery*. There are many details that explain exactly how this service is accomplished, but in a nutshell, IP sends data from point A to point B over an internetwork of connected networks.



**Figure 15-1: The main function of IP: internetwork datagram delivery** IP's overall responsibility is to deliver data between devices on unconnected networks. This figure shows how IP delivers datagrams from one device to another over an internetwork; in this case, a distant client and server communicate with each other by passing IP datagrams over a series of interconnected networks.

**KEY CONCEPT** While the *Internet Protocol* has many functions and characteristics, it can be boiled down to one primary purpose: the delivery of datagrams across an internetwork of connected networks.

Of course, there are many ways in which IP could have been implemented in order to accomplish this task. To understand how the designers of TCP/IP made IP work, let's take a look at the key characteristics used to describe IP and the general manner in which it operates:

**Universally Addressed** In order to send data from point A to point B, it is necessary to ensure that devices know how to identify which device is point B. IP defines the addressing mechanism for the network and uses these addresses for delivery purposes.

**Underlying Protocol-Independent** IP is designed to allow the transmission of data across any type of underlying network that is designed to work with a TCP/IP stack. It includes provisions that allow it to adapt to the requirements of various lower-level protocols such as Ethernet or IEEE 802.11. IP can also run on the special data link protocols, Serial Line Interface Protocol (SLIP) and Point-to-Point Protocol (PPP), that were created for it (see Part II-1). An important example is IP's ability to fragment large blocks of data into smaller ones in order to match the size limitations of physical networks, and then have the recipient reassemble the pieces again as needed.

**Connectionless Delivery** IP is a *connectionless protocol*. This means that when point A wants to send data to point B, it doesn't first set up a connection to point B and then send the data—it just makes the datagram and sends it. (See the section in Chapter 1 on connection-oriented and connectionless protocols for more information on this.)

**Unreliable Delivery** IP is said to be an unreliable protocol. That doesn't mean that one day your IP software will decide to go fishing rather than run your network. It does mean that when datagrams are sent from Device A to Device B, Device A just sends each one and then moves on to the next. IP doesn't keep track of the ones it sent. It does not provide reliability or service-quality capabilities, such as error protection for the data it sends (though it does on the IP header), flow control, or retransmission of lost datagrams. For this reason, IP is sometimes called a *best-effort* protocol. It does what it can to get data to where it needs to go, but makes no guarantees that the data will actually get there.

**Unacknowledged Delivery** Corresponding with its unreliable nature, IP doesn't use acknowledgements. When Device B gets a datagram from Device A, it doesn't send back a “thank you note” to tell Device A that the datagram was received. It leaves Device A in the dark, so to speak.

These last three characteristics might be enough to make you cringe, thinking that giving your data to IP would be somewhat like trusting a new car to your 16-year-old son. If you are going to build an entire network around this protocol, why design it so that it works without connections, doesn't guarantee that the data will get there, and has no means of acknowledging receipt of data?

The reason is simple: Establishing connections, guaranteeing delivery, error checking, and similar insurance-type functions have a cost in *performance*. It takes time, computer resources, and network bandwidth to perform these tasks, and they aren't always necessary for every application. Now, consider that IP carries pretty much *all* user traffic on a TCP/IP network. To build this complexity into IP would burden all traffic with this overhead, whether or not it was needed.

The solution taken by the designers of TCP/IP was to exploit the power of layering. If service-quality features such as connections, error checking, or guaranteed delivery are required by an application, they are provided at the transport layer (or possibly, the application layer). On the other hand, applications that don't need these features can avoid using them. This is the major distinction between the two TCP/IP transport layer protocols: TCP and UDP. TCP is full featured but a bit slower than UDP; UDP is spartan in its capabilities, but faster than TCP. This system is really the best of both worlds, and it works.

## IP Functions

The exact number of IP functions depends on where you draw the line between certain activities. For explanatory purposes, however, I view IP as having four basic functions (or more accurately, function sets):

**Addressing** Before it can deliver datagrams, IP must know where to deliver them. For this reason, IP includes a mechanism for host addressing. Furthermore, since IP operates over internetworks, its system is designed to allow for the unique addressing of devices across arbitrarily large networks. It also contains a structure to facilitate the routing of datagrams to distant networks, if that is required. Since most of the other TCP/IP protocols use IP, an understanding the IP addressing scheme is of vital importance to comprehending much of what goes on in TCP/IP. It is explored fully in Chapters 16 through 20.

**Data Encapsulation and Formatting/Packaging** As the TCP/IP network layer protocol, IP accepts data from the transport layer protocols UDP and TCP. It then encapsulates this data into an IP datagram using a special format prior to transmission.

**Fragmentation and Reassembly** IP datagrams are passed down to the data link layer for transmission on the local network. However, the maximum frame size of each physical and data link network using IP may be different. For this reason, IP includes the ability to *fragment* IP datagrams into pieces, so that they can each be carried on the local network. The receiving device uses the *reassembly* function to re-create the whole IP datagram. Some people view fragmentation and reassembly as distinct functions, though clearly they are complementary, and I view them as being part of the same job.

**Routing and Indirect Delivery** When an IP datagram must be sent to a destination on the same local network, you can do this easily with the network's underlying local area network (LAN), wireless LAN (WLAN), or wide area network (WAN) protocol, using what is sometimes called *direct delivery*. However, in many (if not most cases) the final destination is on a distant network that isn't directly attached

to the source. In this situation, the datagram must be delivered indirectly. This is accomplished by routing the datagram through intermediate devices (*routers*). IP accomplishes this in concert with support from the other protocols including the Internet Control Message Protocol (ICMP) and the TCP/IP gateway/routing protocols such as the Routing Information Protocol (RIP) and the Border Gateway Protocol (BGP).

## IP History, Standards, Versions, and Closely Related Protocols

Since IP is really the architectural foundation for the entire TCP/IP protocol suite, you might have expected that it was created first, and that the other protocols were built upon it. That's usually how you build a structure, after all! The history of IP, however, is a bit more complex. The functions it *performs* were defined at the birth of the protocol, but IP itself didn't exist for the first few years that the protocol suite was being defined.

I explore the early days of TCP/IP in Chapter 8, which provides an overview of the suite as a whole. What is notable about the development of IP is that its functions were originally part of TCP. As a formal protocol, IP was born when an early version of TCP developed in the 1970s for predecessors of the modern Internet was split into TCP at layer 4 and IP at layer 3. The key milestone in the development of IP was the publication of RFC 791, “Internet Protocol,” in September 1981. This standard, a revision of the similar RFC 760 of the previous year, defined the core functionality and characteristics of the version of IP that has been in widespread use for the last two decades.

### IP Versions and Version Numbers

The IP defined in RFC 791 was the first widely used version of IP. Interestingly, however, it is not version 1 of IP but version 4! This would of course imply that there were earlier versions of the protocol at one point. Interestingly, however, there really weren’t. IP was created when its functions were split out from an early version of TCP that combined both TCP and IP functions. TCP evolved through three earlier versions and was split into TCP and IP for version 4. That version number was applied to both TCP and IP for consistency.

**KEY CONCEPT** Version 4 of the *Internet Protocol* (IP) is actually the first version that was widely deployed and is currently the one in widespread use.

So, when you use IP today, you are using IP version 4, which is frequently abbreviated IPv4. Unless otherwise qualified, it’s safe to assume that *IP* means IP version 4—at least for the next few years. (This version number is carried in the appropriate field of all IP datagrams, as described in the topic discussing the IP datagram format in Chapter 21.)

Given that it was originally designed for an internetwork a tiny fraction of the size of our current Internet, IPv4 has proven itself remarkably capable. Various additions and changes have been made over time to how IP is used, especially with respect to addressing, but the core protocol is basically what it was in the early

1980s. There's good reason for this. Changing something as fundamental as IP requires a great deal of development effort and also introduces complexities during transition.

IPv4 has served us well, but people understood that, for various reasons, a new version of IP would eventually be required. Due to the difficulties associated with making such an important change, development of this new version of IP has actually been under way since the mid-1990s. This new version of IP is formally called *Internet Protocol version 6 (IPv6)* and also sometimes referred to as *IP Next Generation* or *IPng*. I discuss the reasons why IPv6 was developed and how it differs from IPv4 in considerable detail in Part II-4 of this book.

A natural question at this point is, "What happened to version 5 of IP?" The answer is that it doesn't exist. While this may seem confusing, version 5 was in fact intentionally skipped in order to *avoid* confusion, or at least to rectify it. The problem with version 5 relates to an experimental TCP/IP protocol called the *Internet Stream Protocol, version 2*, originally defined in RFC 1190. This protocol was originally seen by some as being a peer of IP at the Internet layer in the TCP/IP architecture, and in its standard version, these packets were assigned IP version 5 to differentiate them from normal IP packets (version 4). This protocol apparently never went anywhere, but to be absolutely sure that there would be no confusion, version 5 was skipped over in favor of version 6.

## **IP-Related Protocols**

In addition to the old and new versions of IP, there are several protocols that are *IP-related*. These are protocols that add to or expand on the capabilities of IP functions for special circumstances, but they are not part of IP proper. These are as follows:

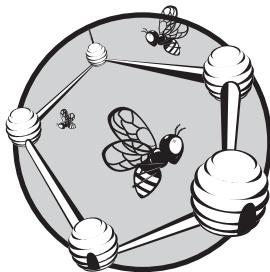
**IP Network Address Translation (IP NAT or NAT)** This protocol provides IP address translation capabilities that allow private networks to be interfaced to public networks in a flexible manner. It allows public IP addresses to be shared and improves security by making it more difficult for hosts on the public network to gain unauthorized access to hosts. It is commonly called *NAT*. This protocol is discussed in Chapter 28.

**IP Security (IPsec)** IPsec defines a set of subprotocols that provide a mechanism for the secure transfer of data using IP. It is rapidly growing in popularity as a security protocol that enables virtual private networks. This protocol is discussed in Chapter 29.

**Mobile IP** This is a protocol that addresses some of the difficulties associated with using IP on computers that frequently move from one network to another. It provides a mechanism that allows data to be automatically routed to a mobile host (such as a notebook computer), without requiring a constant reconfiguration of the device's IP address. This protocol is discussed in Chapter 30.

# 16

## **IPV4 ADDRESSING CONCEPTS AND ISSUES**



The primary job of the Internet Protocol (IP) is delivering messages between devices, and like any good delivery service, it can't do its job too well if it doesn't know where the recipients are located. Obviously then, one of the most important functions of IP is *addressing*. IP addressing is used not only to uniquely identify IP addresses, but also to facilitate the routing of IP datagrams over internetworks. IP addresses are used and referred to extensively in TCP/IP networking.

Even though the original IP addressing scheme was relatively simple, it has become complex over time as changes have been made to it to allow it to deal with various addressing requirements. The more advanced styles of IP addressing, such as subnetting and classless addressing, are the ones used most in modern networks. However, they can be a bit confusing to understand. To help make sense of them, we must start at the beginning with a discussion of the fundamentals of IP addressing.

In this chapter, I begin a larger exploration of IP addressing by explaining the key concepts and issues behind it. I begin with an overview of IP addressing and a discussion of what it is all about. I describe the size of IP addresses, the concept of its address space, and the notation usually used for IP addresses. I provide basic information on the structure of an IP address and how it is divided into a network identifier and host identifier. I then describe the different types of IP addresses and the additional information, such as a subnet mask and default gateway, that often accompanies an IP address on larger networks. I provide a brief description of how multiple addresses are sometimes assigned to single devices and why. I conclude with a description of the process by which public IP addresses are registered and managed, and the organizations that do this work for the global Internet.

**BACKGROUND INFORMATION** *If you are not familiar with at least the basics of how binary numbers work, and also with how to convert between binary and decimal numbers, I recommend reading Chapter 4, which provides some background on data representation and the mathematics of computing, before you proceed here.*

## IP Addressing Overview and Fundamentals

IP addressing is important because it facilitates the primary function of the IP: the delivery of datagrams across an internetwork. When you examine this in more detail, it becomes apparent that the IP address actually has two different functions, as follows:

**Network Interface Identification** Like a street address, the IP address provides unique identification of the interface between a device and the network. This is required to ensure that the datagram is delivered to the correct recipients.

**Routing** When the source and destination of an IP datagram are not on the same network, the datagram must be delivered indirectly using intermediate systems. This is a process called *routing*. The IP address is an essential part of the system used to route datagrams.

You may have noticed a couple of things about this short list. One is that I said the IP address identifies the *network interface*, not that it identifies the *device* itself. This distinction is important because it underscores the concept that IP is oriented around connections to a large, virtual network at layer 3, which can span multiple physical networks. Some devices, such as routers, will have more than one network connection, necessary to take datagrams from one network and route them onto another. This means they will also have more than one IP address—one per connection.

You might also find it curious that I said that the IP address facilitates routing. How can it do that? The answer is that the addressing system is designed with a structure that can be interpreted to allow routers to determine what to do with a datagram based on the values in the address. Numbers related to the IP address, such as the subnet mask when subnetting is used, support this function.

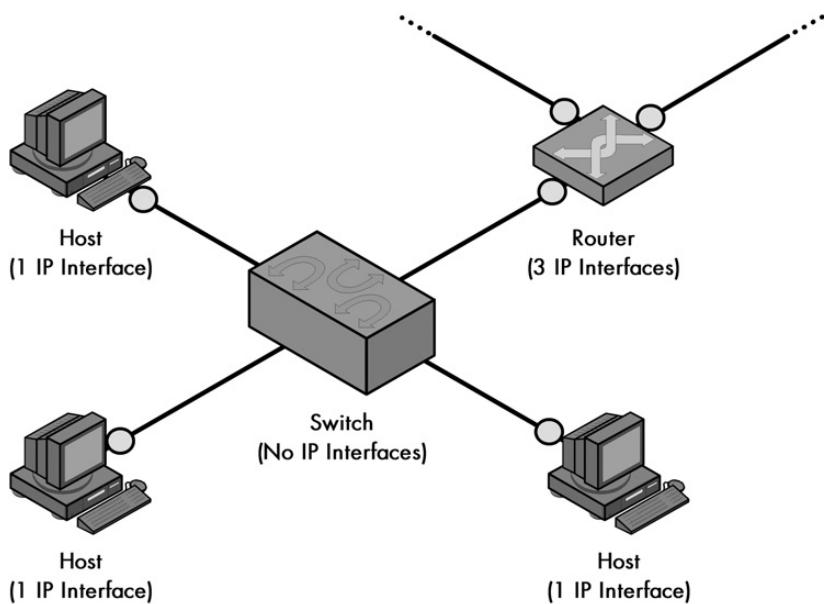
Let's look at some of the more important issues and characteristics associated with IP addresses in general terms.

## **Number of IP Addresses Per Device**

Any device that has data sent to it at the network layer will have at least one IP address: one per network interface. This means that normal hosts such as computers and network-capable printers usually get one IP address, while routers get more than one IP address. Some special hosts may have more than one IP address if they are multihomed—connected to more than one network.

Lower-level network interconnection devices—such as repeaters, bridges, and switches—don’t require an IP address because they pass traffic based on layer 2 (data link layer) addresses. Network segments connected by bridges and switches form a single broadcast domain, and any devices on them can send data to each other directly without routing. To IP, these devices are essentially invisible; they are no more significant than the wires that connect devices together (with a couple of exceptions). Such devices may, however, optionally have an IP address for management purposes. In this regard, they are acting like a regular host on the network.

Figure 16-1 shows the IP interfaces of a few common LAN devices as small circles. Each regular host has one interface, while the router that serves this LAN has three, since it connects to three different networks. Note that the LAN switch has no IP interfaces; it connects the hosts and router at layer 2. (Also see Figure 16-5, which shows the IP interfaces of devices in a more complex configuration.)



**Figure 16-1: IP interfaces for common network devices** Regular hosts have one interface; routers usually have more than one; and switches have none (because they operate at layer 2).

## **Address Uniqueness and Network Specificity**

Each IP address on a single internetwork must be unique. (This seems rather obvious, although there are exceptions in IPv6, in the form of special anycast addresses, as discussed in Chapter 25.)

Since IP addresses represent network interfaces and are used for routing, the IP address is specific to the network to which it is connected. If the device moves to a new network, the IP address will usually have to change as well. For the full reason why, see the discussion of basic IP address structure later in this chapter. This issue was a primary motivation for the creation of Mobile IP (covered in Chapter 30).

### **Contrasting IP Addresses and Data Link Layer Addresses**

IP addresses are used for network-layer data delivery across an internetwork. This makes IP addresses quite different from the data link layer address of a device, such as its Ethernet MAC address. (In TCP/IP parlance, these are sometimes called *physical addresses* or *hardware addresses*.)

At the network layer, a single datagram may be sent from Device A to Device B. However, the actual delivery of the datagram may require that it passes through a dozen or more physical devices if Device A and Device B are not on the same network.

It is also necessary to provide a function that maps between IP and data link layer addresses. In TCP/IP, this is the job of the Address Resolution Protocol (ARP; see Chapter 13).

In a physical network such as an Ethernet, the MAC address is all the information needed to send data between devices. In contrast, an IP address represents only the final delivery point of the datagram. The route taken depends on the characteristics of the network paths between the source and destination devices. It is even possible that there may not be a route between any two devices, which means two devices cannot exchange data, even if they know each other's addresses!

### **Private and Public IP Network Addresses**

There are two distinct ways that a network can be set up with IP addresses. On a *private network*, a single organization controls the assignment of the addresses for all devices; they have pretty much absolute control to do what they wish in selecting numbers, as long as each address is unique.

In contrast, on a *public network*, a mechanism is required to ensure that organizations don't use overlapping addresses and that they enable efficient routing of data between organizations. The best-known example of this is the Internet, where public IP registration and management facilities have been created to address this issue. There are also advanced techniques now, such as IP Network Address Translation (NAT), which allow a network using private addresses to be interfaced to a public TCP/IP network.

### **IP Address Configuration and Addressing Types**

IP addresses can be set up as either a static or dynamic configuration. In a *static configuration* setup, each device is manually configured with an IP address that doesn't change. This is fine for small networks but quickly becomes an administrative nightmare in larger networks, when changes are required. The alternative,

*dynamic configuration*, allows IP addresses to be assigned to devices and changed under software control. The two host configuration protocols, BOOTP and DHCP, were created to fill this latter function (see Part III-3).

Additionally, provision is included in the IP addressing scheme for all three basic types of addressing: unicast, multicast, and broadcast.

**KEY CONCEPT** IP addresses serve the dual function of device identification and routing. Each network interface requires one IP address, which is network specific. IP addresses can be either statically or dynamically allocated, and come in unicast, multicast, and broadcast forms.

## IP Address Size, Address Space, and Notation

Now that you have looked at the general issues and characteristics associated with IP addresses, it's time to get past the introductions and dig into the "meat" of the IP address discussion. Let's start by looking at the physical construction and size of the IP address and how it is referred to and used.

### IP Address Size and Binary Notation

At its simplest, the IP address is just a 32-bit binary number: a set of 32 ones or zeros. At their lowest levels, computers always work in binary, and this also applies to networking hardware and software. While different meanings are ascribed to different bits in the address, the address itself is just a 32-digit binary number.

People don't work too well with binary numbers, because they are long and complicated, and the use of only two digits makes them hard to differentiate. (Quick, which of these is larger: 11100011010100101001100110110001 or 11100011010100101001101110110001?) For this reason, when you use IP addresses, you don't work with them in binary except when absolutely necessary.

The first thing that people would naturally do with a long string of bits is to split it into four eight-bit octets (or bytes, even though the two aren't technically the same; see Chapter 4), to make it more manageable. So 11100011010100101001101110110001 would become 11100011 - 01010010 - 10011101 - 10110001. Then you could convert each of those octets into a more manageable two-digit hexadecimal number to yield the following: E3 - 52 - 9D - B1. This is, in fact, the notation used for IEEE 802 MAC addresses, except that they are 48 bits long, so they have six two-digit hex numbers, and they are usually separated by colons, not dashes, as I used here.

(Incidentally, the second binary number is the larger one.)

### IP Address Dotted Decimal Notation

Most people still find hexadecimal a bit difficult to work with. So, IP addresses are normally expressed with each octet of eight bits converted to a decimal number and the octets separated by a period (a *dot*). Thus, the previous example would become 227.82.157.177, as shown in Figure 16-2. This is usually called *dotted decimal notation* for rather obvious reasons. Each of the octets in an IP address can take on the values from 0 to 255, so the lowest value is theoretically 0.0.0.0 and the highest is 255.255.255.255.

**KEY CONCEPT** IP addresses are 32-bit binary numbers, which can be expressed in binary, hexadecimal, or decimal form. Most commonly, they are expressed by dividing the 32 bits into four bytes and converting each to decimal, then separating these numbers with dots to create dotted decimal notation.

	0	8	16	24	32
Binary	11100011	01010010	10011101	10110001	
Hexadecimal	E3	52	9D	B1	
Dotted Decimal	227	82	157	177	

**Figure 16-2: IP address binary, hexadecimal, and dotted decimal representations** The binary, hexadecimal, and decimal representations of an IP address are all equivalent.

Dotted decimal notation provides a convenient way to work with IP addresses when communicating among people. Never forget that to the computers, the IP address is always a 32-bit binary number; you'll understand the importance of this when you look at how the IP address is logically divided into components in the next topic, and when you examine techniques that manipulate IP addresses, such as subnetting.

## IP Address Space

Since the IP address is 32 bits wide, this provides a theoretical *address space* of  $2^{32}$ , or 4,294,967,296 addresses. This seems like quite a lot of addresses, and in some ways, it is. However, as you will see, due to how IP addresses are structured and allocated, not every one of those addresses can actually be used.

One of the unfortunate legacies of the fact that IP was originally created on a rather small internetwork is that decisions were made that wasted much of the address space. For example, all IP addresses starting with 127 in the first octet are reserved for the loopback function. Just this one decision makes 1/256th of the total number, or 16,277,216 addresses, no longer available. There are also other ways that the IP address space was not conserved. This caused difficulty as the Internet grew in size. (You'll see more about this in Chapter 17, which covers classful addressing.)

**KEY CONCEPT** Since IP addresses are 32 bits long, the total address space of IPv4 is  $2^{32}$  or 4,294,967,296 addresses. However, not all of these addresses can be used, for a variety of reasons.

This IP address space dictates the limit on the number of addressable interfaces in *each* IP internetwork. So, if you have a private network, you can, in theory, have four-billion-plus addresses. However, in a public network such as the Internet, all

devices must share the available address space. Techniques such as Classless Inter-Domain Routing (CIDR), or supernetting, and NAT were designed in part to utilize the existing Internet IP address space more efficiently. IPv6 expands the IP address size from 32 bits all the way up to 128, which increases the address space to a ridiculously large number and makes the entire matter of address space size moot.

## IP Basic Address Structure and Main Components

As I mentioned in the IP addressing overview, one of the ways that IP addresses are used is to facilitate the routing of datagrams in an IP internetwork. This is made possible because of the way that IP addresses are structured and how that structure is interpreted by network routers.

### **Network ID and Host ID**

As you just saw, each IPv4 address is 32 bits long. When you refer to the IP address, you use a dotted decimal notation, while the computer converts this into binary. However, even though these sets of 32 bits are considered a single entity, they have an internal structure containing two components:

**Network Identifier (Network ID)** A certain number of bits, starting from the left-most bit, is used to identify the network where the host or other network interface is located. This is also sometimes called the *network prefix* or even just the *prefix*.

**Host Identifier (Host ID)** The remainder of the bits is used to identify the host on the network.

**NOTE** *By convention, IP devices are often called hosts for simplicity, as I do throughout this book. Even though each host usually has a single IP address, you should remember that IP addresses are strictly associated with network layer network interfaces, not physical devices, and a device may therefore have more than one IP address (especially a router or multihomed host).*

As you can see in Figure 16-3, this really is a fairly simple concept. The fundamental division of the bits of an IP address is into a network ID and host ID. In this illustration, the network ID is 8 bits long, and the host ID is 24 bits in length. This is similar to the structure used for phone numbers in North America. The telephone number (401) 555-7777 is a ten-digit number that's usually referred to as a single phone number. However, it has a structure. In particular, it has an area code (401) and a local number (555-7777).

The fact that the network ID is contained in the IP address is what partially facilitates the routing of IP datagrams when the address is known. Routers look at the network portion of the IP address to first determine if the destination IP address is on the same network as the host IP address. Then routing decisions are made based on information the routers keep about where various networks are located. Again, this is conceptually similar to how the area code is used by the equivalent of routers in the phone network to switch telephone calls. The host portion of the address is used by devices on the local portion of the network.

	0	8	16	24	32
Binary	11100011	01010010	10011101	10110001	
Dotted Decimal	227	82	157	177	
Network ID				Host ID	

IP Address: 227.82.157.177

**Figure 16-3: Basic IP address division: network ID and host ID** This diagram shows one of the many ways to divide an IP address into a network ID and host ID.

### Location of the Division Between Network ID and Host ID

One difference between IP addresses and phone numbers is that the dividing point between the bits used to identify the network and those that identify the host isn't fixed. It depends on the nature of the address, the type of addressing being used, and other factors.

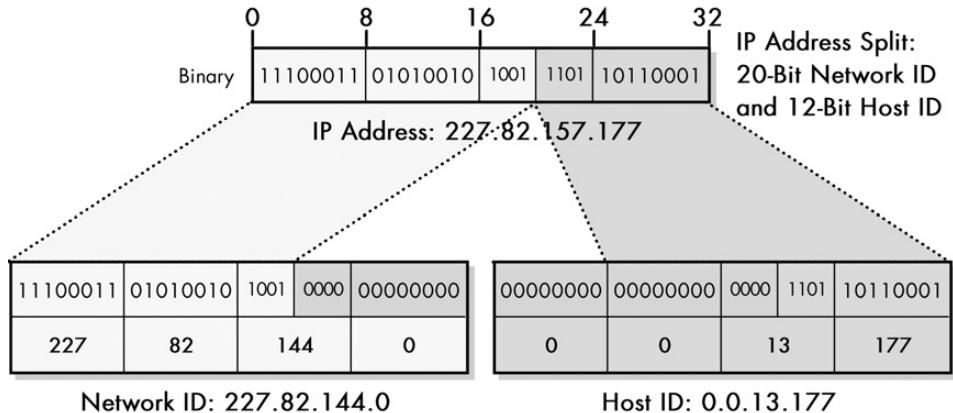
Take the previous example of 227.82.157.177 (see Figure 16-2). It is possible to divide this into a network ID of 227.82 and a host ID of 157.177. Alternatively, the network ID might be 227 and the host ID might be 82.157.177 within that network.

To express the network and host IDs as 32-bit addresses, you add zeros to replace the missing pieces. With a network ID of 227 and a host ID of 82.157.177, the address of the network becomes 227.0.0.0 and the address of the host 0.82.157.177. (In practice, network addresses of this sort are routinely seen with the added zeros; network IDs are not seen as often in 32-bit form this way.)

Lest you think from these examples that the division must always be between whole octets of the address, you should know that it's also possible to divide it in the middle of an octet. For example, you could split the IP address 227.82.157.177 so that there were 20 bits for the network ID and 12 bits for the host ID. The process is the same, but determining the dotted decimal ID values is more tricky because here, the 157 is split into two binary numbers. The results are 227.82.144.0 for the network ID and 0.0.0.13.177 for the host ID, as shown in Figure 16-4.

Since IP addresses are normally expressed as four dotted-decimal numbers, educational resources often show the division between the network ID and host ID occurring on an octet boundary. However, it's essential to remember that the dividing point often appears in the middle of one of these eight-bit numbers. In Figure 16-4, the network ID is 20 bits long, and the host ID 12 bits long. This results in the third number of the original IP address, 157, being split into 144 and 13.

The place where the line is drawn between the network ID and the host ID must be known in order for devices such as routers to know how to interpret the address. This information is conveyed either implicitly or explicitly, depending on the type of IP addressing in use, as I discuss next.



**Figure 16-4: Mid-octet IP address division** IP addresses need not be divided between network ID and host ID on octet boundaries. The division here is into a 20-bit network ID and a 12-bit host ID.

**KEY CONCEPT** The basic structure of an IP address consists of two components: the network ID and host ID. The dividing point of the 32-bit address is not fixed, but depends on a number of factors and can occur in a variety of places, including in the middle of a dotted-decimal octet.

Since the IP address can be split into network ID and host ID components, it is also possible to use either one or the other by itself, depending on context. These addresses are assigned special meanings. For example, if the network ID is used with all ones as the host ID, this indicates a broadcast to the entire network. Similarly, if the host ID is used by itself with all zeros for the network ID, this implies an IP address sent to the host of that ID on the local network, whatever that might be. This is explained in much more detail in Chapter 17.

It is the inclusion of the network ID in the IP address of each host on the network that causes the IP addresses to be network-specific. If you move a device from one network to a different one, the network ID must change to that of the new network. Therefore, the IP address must change as well. This is an unfortunate drawback that shows up most commonly when dealing with mobile devices; see Chapter 30.

## IP Addressing Categories and IP Address Adjuncts

We just explored how the 32 bits in an IP address are fundamentally divided into the network ID and host ID. The network ID is used for routing purposes, and the host ID uniquely identifies each network interface on the network. In order for devices to know how to use IP addresses on the network, they must be able to tell which bits are used for each ID. However, the dividing line is not predefined. It depends on the type of addressing used in the network.

Understanding how these IDs are determined leads us into a larger discussion of the three main categories of IP addressing schemes: classful, subnetted, and classless. Each of these uses a slightly different system of indicating where in the IP address the host ID is found.

### ***Conventional (Classful) Addressing***

The original IP addressing scheme is set up so that the dividing line occurs only in one of a few locations: on octet boundaries. Three main classes of addresses—A, B, and C—are differentiated based on how many octets are used for the network ID and how many for the host ID. For example, Class C addresses devote 24 bits to the network ID and 8 bits to the host ID. This type of addressing is now often referred to by the made-up word *classful* to differentiate it from the newer classless scheme.

This most basic addressing type uses the simplest method to divide the network and host IDs: The class, and therefore the dividing point, are encoded into the first few bits of each address. Routers can tell from these bits which octets belong to which identifier.

### ***Subnetted Classful Addressing***

In the subnet addressing system, the two-tier network and host division of the IP address is made into a three-tier system by taking some number of bits from a Class A, B, or C host ID and using them for a *subnet identifier (subnet ID)*. The network ID is unchanged. The subnet ID is used for routing within the different subnetworks that constitute a complete network, thereby providing extra flexibility for administrators. For example, consider a Class C address that normally uses the first 24 bits for the network ID and remaining 8 bits for the host ID. The host ID can be split into, say, 3 bits for a subnet ID and 5 bits for the host ID.

This system is based on the original classful scheme, so the dividing line between the network ID and full host ID is based on the first few bits of the address as before. The dividing line between the subnet ID and the “subhost” ID is indicated by a 32-bit number called a *subnet mask*. In the previous example, the subnet mask would be 27 ones followed by 5 zeros—the zeros indicate what part of the address is the host. In dotted decimal notation, this would be 255.255.255.224.

### ***Classless Addressing***

In the classless system, the classes of the original IP addressing scheme are tossed out the window. The division between the network ID and host ID can occur at an arbitrary point, not just on octet boundaries, as in the classful scheme.

The dividing point is indicated by putting the number of bits used for the network ID, called the *prefix length*, after the address. (Recall that the network ID bits are also sometimes called the *network prefix*, so the network ID size is the prefix length.) For example, if 227.82.157.177 is part of a network where the first 27 bits are used for the network ID, that network would be specified as 227.82.157.160/27. The /27 is conceptually the same as the 255.255.255.224 subnet mask, since it has 27 one bits followed by 5 zeros.

**KEY CONCEPT** An essential factor in determining how an IP address is interpreted is the addressing scheme in which it is used. The three methods, arranged in increasing order of age, complexity, and flexibility, are classful addressing, subnetted classful addressing, and classless addressing.

This introduction to the concepts of classful, subnetted, and classless addressing was designed to show you how they impact the way the IP address is interpreted. I have greatly summarized important concepts here. All three methods are explained in their own chapters in full detail.

### **Subnet Mask and Default Gateway**

In the original classful scheme, the division between network ID and host ID is implied. However, if either subnetting or classless addressing is used, then the *subnet mask* (or *slash number*, which is equivalent) is required to fully qualify the address. These numbers are considered adjuncts to the IP address and usually mentioned with the address itself, because without them, it is not possible to know where the network ID ends and the host ID begins.

One other number that is often specified along with the IP address for a device is the *default gateway* identifier. In simplest terms, this is the IP address of the router that provides default routing functions for a particular device. When a device on an IP network wants to send a datagram to a device it can't see on its local IP network, it sends it to the default gateway, which takes care of routing functions. Without this, each IP device would also need to have knowledge of routing functions and routes, which would be inefficient. See Chapter 23, which discusses IP routing concepts, and Chapter 37 through 41, which cover TCP/IP routing protocols, for more information.

## **Number of IP Addresses and Multihoming**

Each network interface on an IP internetwork has a separate IP address. In a classic network, each regular computer, usually called a *host*, attaches to the network in exactly only one place, so it will have only one IP address. This is what most of us are familiar with when using an IP network (and is also why most people use the term *host* when they really mean *network interface*).

If a device has more than one interface to the internetwork, it will have more than one IP address. The most obvious case where this occurs is with routers, which connect together different networks and thus must have an IP address for the interface on each one. It is also possible for hosts to have more than one IP address, however. Such a device is sometimes said to be *multihomed*.

There are two ways that a host can be multihomed:

**Two or More Interfaces to the Same Network** Devices such as servers or high-powered workstations may be equipped with two physical interfaces to the same network for performance and reliability reasons. They will have two IP addresses on the same network with the same network ID.

**Interfaces to Two or More Different Networks** Devices may have multiple interfaces to different networks. The IP addresses will typically have different network IDs in them.

Figure 16-5 shows examples of both types of multihomed device. Of course, these could be combined, with a host having two connections to one network and a third to another network. There are also some other special cases, such as a host with a single network connection having multiple IP address aliases.

**NOTE** When subnetting is used, the same distinction can be made between multihoming to the same subnet or a different subnet.

Now, let's consider the second case. If a host has interfaces to two or more different networks, could it pass IP datagrams between them? Yes, if it had the right software running on it. And wouldn't that make the host a router, of sorts? In fact, that is exactly the case. A multihomed host with interfaces to two networks can use software to function as a router. This is sometimes called *software routing*.

Using a host as a router has certain advantages and disadvantages compared to a hardware router. A server that is multihomed can perform routing functions and also, well, act as a server. A dedicated hardware router is designed for the job of routing and usually will be more efficient than a software program running on a host.

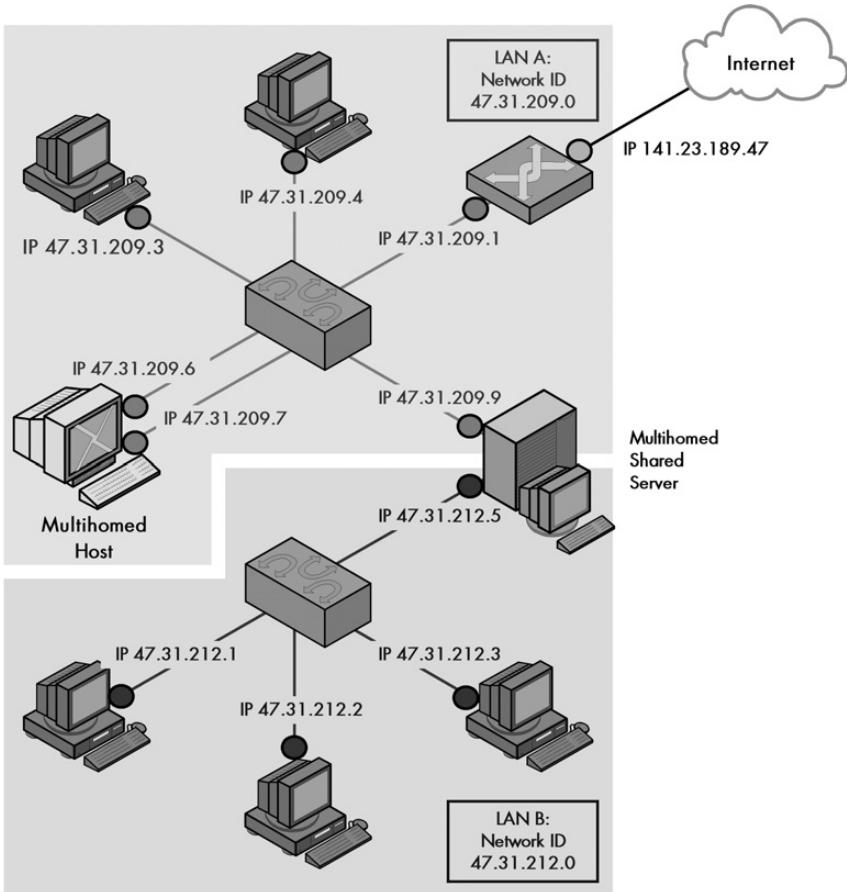
**KEY CONCEPT** A host with more than one IP network interface is said to be multihomed. A multihomed device can have multiple connections to the same network, to different networks, or both. A host connected to two networks can be configured to function as a router.

Multihoming was once considered a fairly esoteric application, but has become more common in recent years. This is also true of multihoming on different networks for software routing use. In fact, you may be doing this in your home without realizing it.

Suppose you have two PCs networked together and a single phone line to connect to the Internet. One computer dials up to your Internet service provider (ISP) and runs software such as Microsoft's Internet Connection Sharing (ICS) to let the other computer access the Internet. Millions of people do this every day—they have a multihomed system (the one connecting to the Internet and the other PC) with ICS acting in the role of a software router (though there are some technical differences between ICS and a true router, of course).

## IP Address Management and Assignment Methods and Authorities

What would happen if you told someone that you lived at 34 Elm Street, and when he turned onto your road, he found four different houses with the number 34 on them? He probably would find your place eventually but wouldn't be too pleased. Neither would you or your mail carrier! And all of you folks are much smarter than computers. Like street addresses, IP addresses must be unique for them to be useful.



**Figure 16-5: Multihomed devices on an IP internetwork** This internetwork consists of two LANs, A (above) and B (below). LAN A has a multihomed workstation, shown with two IP network interface “circles.” The two LANs are connected together through a multihomed, shared server that has been configured to route traffic between them. Note that this server also handles all traffic passing between LAN B and the Internet (since the Internet connection is in LAN A only).

Since IP datagrams are sent only within the confines of the IP internetwork, they must be unique within each internetwork. If you are a company with your own private internetwork, this isn’t really a big problem. Whoever is in charge of maintaining the internetwork keeps a list of what numbers have been used where and makes sure that no two devices are given the same address. However, what happens in a public network with many different organizations? Here, it is essential that the IP address space be managed across the organizations to ensure that they use different addresses. It’s not feasible to have each organization coordinate its activities with each other one. Therefore, some sort of centralized *management authority* is required.

At the same time that you need someone to ensure that there are no conflicts in address assignment, you don’t want users of the network to have to go to this central authority every time they need to make a change to their network. It makes more sense to have the authority assign numbers in blocks or chunks to organizations

based on the number of devices they want to connect to the network. The organizations can manage those blocks as they see fit, and the authority's job is made easier because it deals in blocks instead of billions of individual addresses and machines.

The Internet, as the big IP internetwork, requires this coordination task to be performed for millions of organizations worldwide. The job of managing IP address assignment on the Internet was originally carried out by a single organization: the *Internet Assigned Number Authority* (IANA). IANA was responsible for allocating IP addresses, along with other important centralized coordination functions such as managing universal parameters used for TCP/IP protocols. In the late 1990s, a new organization called the *Internet Corporation for Assigned Names and Numbers* (ICANN) was created. ICANN now oversees the IP address assignment task of IANA, as well as managing other tasks such as Domain Name System (DNS) name registration (see Chapter 54).

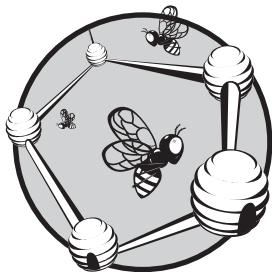
IP addresses were originally allocated directly to organizations. The original IP addressing scheme was based on classes, and so IANA would assign addresses in Class A, B, and C blocks. Today, addressing is classless, using CIDR's hierarchical addressing scheme. IANA doesn't assign addresses directly, but rather delegates them to regional Internet registries (RIRs). These are APNIC, ARIN, LACNIC, and RIPE NCC. Each RIR can, in turn, delegate blocks of addresses to lower-level registries such as national Internet registries (NIRs) and local Internet registries (LIRs).

Eventually, blocks of addresses are obtained by ISPs for distribution to end-user organizations. Some of the ISP's customers are end-user organizations, but others are (smaller) ISPs themselves. They can, in turn, use or delegate the addresses in their blocks. This can continue for several stages in a hierarchical fashion. This arrangement helps ensure that IP addresses are assigned and used in the most efficient manner possible. See Chapter 20, which discusses CIDR, for more information on how this works.

IANA, ICANN, and the RIRs are responsible for more than just IP address allocation, though I have concentrated on IP addresses here for obvious reasons. For more general information on IANA, ICANN, APNIC, ARIN, LACNIC, and RIPE NCC, try a can of alphabet soup—or Chapter 3, which provides an overview of the Internet registration authorities.

# 17

## **CLASSFUL (CONVENTIONAL) ADDRESSING**



The original addressing method for IP addresses divided the IP address space into five chunks of different sizes called *classes*, and assigned blocks of addresses to organizations from these classes based on the size and requirements of the organization. In this classful addressing scheme, each class is reserved for a particular purpose, with the main address classes differentiated based on how many octets are used for the network identifier (network ID) and how many are used for the host identifier (host ID).

In this chapter, I describe classful IP addressing. I begin with an overview of the concept and general description of the different classes. I discuss the network and host IDs and address ranges associated with the different classes. I discuss the capacities of each of the commonly used classes, meaning how many networks belong to each and how many hosts each network can contain. I discuss the special meanings assigned to certain IP address patterns and the special ranges reserved for private IP addressing, loopback functions, and

multicasting. I conclude with a discussion of the problems with this type of addressing, which led to it being abandoned in favor of subnetting, and eventually, classless assignment of the IP address space.

**NOTE** *The classful addressing scheme has been replaced by the classless addressing system described in Chapter 20. However, I think it is still important to understand how this original system operates, as it forms the basis for the more sophisticated addressing mechanisms.*

## IP Classful Addressing Overview and Address Classes

The developers of the Internet Protocol (IP) recognized that organizations come in different sizes and would therefore need varying numbers of IP addresses on the Internet. They devised a system to divide the IP address space into *classes*, each of which contained a portion of the total addresses and was dedicated to specific uses. Some classes would be devoted to large networks on the Internet, while others would be reserved for smaller organizations or special purposes.

This original system had no name; it was simply “the” IP addressing system. Today it is called the *classful addressing scheme* to differentiate it from the newer classless scheme.

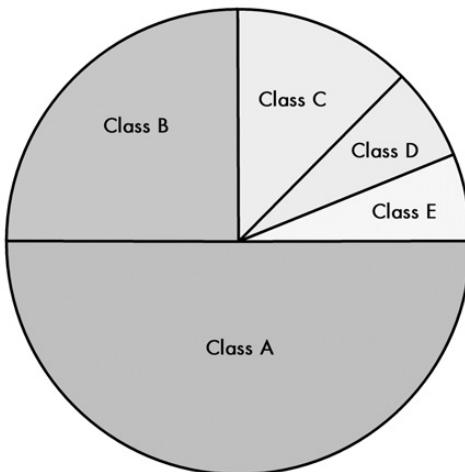
### IP Address Classes

There are five classes in the classful system, which are assigned the letters A through E. Table 17-1 provides some general information about the classes, their intended uses, and their characteristics.

**Table 17-1:** IP Address Classes and Class Characteristics and Uses

IP Address Class	Fraction of Total IP Address Space	Number of Network ID Bits	Number of Host ID Bits	Intended Use
Class A	1/2	8	24	Unicast addressing for very large organizations with hundreds of thousands or millions of hosts to connect to the Internet
Class B	1/4	16	16	Unicast addressing for medium to large organizations with many hundreds to thousands of hosts to connect to the Internet
Class C	1/8	24	8	Unicast addressing for smaller organizations with no more than about 250 hosts to connect to the Internet
Class D	1/16	n/a	n/a	IP multicasting
Class E	1/16	n/a	n/a	Reserved for experimental use

Looking at this table (and Figure 17-1), you can see that Classes A, B, and C take up most of the total address space (seven-eighths of it). These are the classes used for *unicast* IP addressing and messages sent to a single network interface. (The blocks also include associated broadcast addresses for these networks.) This is what I usually consider normal IP addressing.



**Figure 17-1:** Division of IPv4 address space into classes

You can think of Classes A, B, and C as the papa bear, mama bear, and baby bear of traditional IP addressing. They allow the Internet to provide addressing for a small number of very large networks, a moderate number of medium-sized organizations, and a large number of smaller companies. This approximately reflects the distribution of organization sizes in the real world, though the large gulf in the maximum number of hosts allowed for each address class leads to inflexibility, as I will discuss later in the chapter.

As you can see, the classes differ in where they draw the line between the network ID and the host ID portions of the addresses they contain. However, in each case, the division is made on octet boundaries. In classful addressing, the division does not occur within an octet.

Classes D and E are special—to the point where many people don't even realize they exist. Class D is used for IP multicasting, while Class E is reserved for experimental use (by designers of the Internet). I discuss IP multicast addressing later in this chapter.

**KEY CONCEPT** The classful IP addressing scheme divides the IP address space into five classes, A through E, of differing sizes. Classes A, B, and C are the most important ones, designated for conventional unicast addresses and taking up seven-eighths of the address space. Class D is reserved for IP multicasting, and Class E is reserved for experimental use.

### **Rationale for Classful Addressing**

While the drawbacks of the classful system are often discussed today (as you'll see later in this chapter), it's important to keep in context what the size of the Internet was when this system was developed. The Internet was tiny then, and the 32-bit address space seemed enormous by comparison to even the number of machines

its creators envisioned years into the future. It's only fair to also remember the following advantages of the classful system developed over 25 years ago:

**Simplicity and Clarity** There are only a few classes to choose from, and it's very simple to understand how the addresses are split up. The distinction between classes is clear and obvious. The divisions between network ID and host ID in Classes A, B, and C are on octet boundaries, making it easy to tell what the network ID is of any address.

**Reasonable Flexibility** Three levels of granularity match the sizes of large, medium-sized, and small organizations reasonably well. The original system provided enough capacity to handle the anticipated growth rate of the Internet at the time.

**Routing Ease** As you will see shortly, the class of the address is encoded right into the address to make it easy for routers to know what part of any address is the network ID and what part is the host ID. There was no need for adjunct information such as a subnet mask.

**Reserved Addresses** Certain addresses are reserved for special purposes. This includes not just Classes D and E, but also special reserved address ranges for private addressing.

Of course, it turned out that some of the decisions in the original IP addressing scheme were regrettable—but that's the benefit of hindsight. I'm sure we would all like to have back the 268-odd million addresses that were set aside for Class E. While it may seem wasteful now to have reserved a full one-sixteenth of the address space for experimental use, remember that the current size of the Internet was never anticipated even 10 years ago, never mind 25. Furthermore, it's good practice to reserve some portion of any scarce resource for future use.

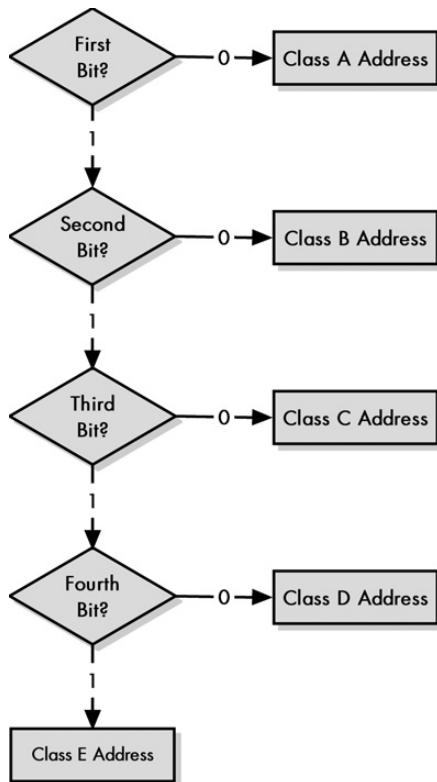
## **IP Classful Addressing Network and Host Identification and Address Ranges**

The classful IP addressing scheme divides the total IP address space into five classes, A through E. One of the benefits of the relatively simple classful scheme is that information about the classes is encoded directly into the IP address. This means you can determine in advance which address ranges belong to each class. It also means the opposite is possible: You can identify which class is associated with any address by examining just a few bits of the address. This latter benefit was one of the main motivators for the initial creation of the classful system.

### ***Classful Addressing Class Determination Algorithm***

When TCP/IP was first created, computer technology was still in its infancy. Routers needed to be able to quickly make decisions about how to move IP datagrams around. The IP address space was split into classes in such a way that, by looking at only the first few bits of any IP address, the router could easily tell how to choose between the network and host ID, and thus what to do with the datagram.

The number of bits the router needs to look at may be as few as one or as many as four, depending on what it finds when it starts looking. The algorithm used to determine the class corresponds to the system used to divide the address space, as illustrated in Figure 17-2.



**Figure 17-2: Class determination algorithm for classful IP addresses** The simplicity of the classful IP addressing can be seen in the very uncomplicated algorithm used to determine the class of an address.

Here are the four very basic steps in the algorithm:

1. If the first bit is a 0, it's a Class A address, and you're done. (Half the address space has a 0 for the first bit, so this is why Class A takes up half the address space.) If it's a 1, continue to step 2.
2. If the second bit is a 0, it's a Class B address, and you're done. (Half of the remaining non-Class A addresses, or one quarter of the total.) If it's a 1, continue to step 3.
3. If the third bit is a 0, it's a Class C address, and you're done. (Half again of what's left, or one-eighth of the total.) If it's a 1, continue to step 4.
4. If the fourth bit is a 0, it's a Class D address. (Half the remainder, or one-sixteenth of the address space.) If it's a 1, it's a Class E address. (The other half, one-sixteenth.)

And that's pretty much it.

## Determining Address Class from the First Octet Bit Pattern

As humans, of course, we generally work with addresses in dotted decimal notation and not in binary, but it's pretty easy to see the ranges that correspond to the classes. For example, consider Class B. The first two bits of the first octet are 10. The remaining bits can be any combination of ones and zeros. This is normally represented as 10xx xxxx (shown as two groups of four for readability). Thus, the binary for the first octet can range from **1000 0000** to **1011 1111** (128 to 191 in decimal). So in the classful scheme, any IP address whose first octet is between 128 and 191 inclusive is a Class B address.

Table 17-2 shows the bit patterns for each of the five classes and the way that the first octet ranges can be calculated. The first column shows the format of the first octet of the IP address; the *xs* can be either a zero or a one. Next are the lowest and highest value columns for each class in binary (the fixed few bits are in bold print so you can see that they do not change while the others do), followed by the corresponding range for the first octet, in decimal.

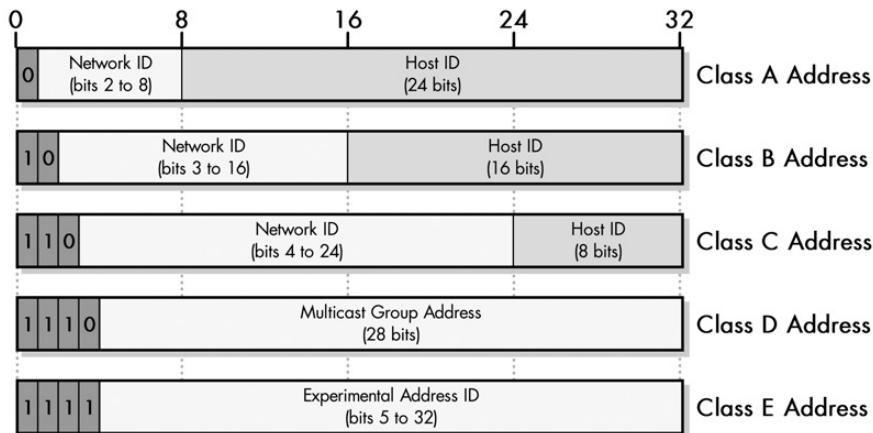
**Table 17-2:** IP Address Class Bit Patterns, First-Octet Ranges, and Address Ranges

IP Address Class	First Octet of IP Address	Lowest Value of First Octet (Binary)	Highest Value of First Octet (Binary)	Range of First Octet Values (Decimal)	Octets in Network ID/Host ID	Theoretical IP Address Range
Class A	0xxx xxxx	0000 0001	0111 1110	1 to 126	1 / 3	1.0.0.0 to 126.255.255.255
Class B	10xx xxxx	1000 0000	1011 1111	128 to 191	2 / 2	128.0.0.0 to 191.255.255.255
Class C	110x xxxx	1100 0000	1101 1111	192 to 223	3 / 1	192.0.0.0 to 223.255.255.255
Class D	1110 xxxx	1110 0000	1110 1111	224 to 239	—	224.0.0.0 to 239.255.255.255
Class E	1111 xxxx	1111 0000	1111 1111	240 to 255	—	240.0.0.0 to 255.255.255.255

This table also shows the *theoretical* lowest and highest IP address ranges for each of the classes. This means that they are the result of taking the full span of binary numbers possible in each class. In reality, some of the values are not available for normal use. For example, even though the range 192.0.0.0 to 192.0.0.255 is technically in Class C, it is reserved and not actually used by hosts on the Internet.

Also, certain IP addresses cannot be used because they have special meaning. For example, 255.255.255.255 is a reserved broadcast address. In a similar vein, note that the range for Class A is from 1 to 126 and not 0 to 127 as you might have expected. This is because Class A networks 0 and 127 are reserved; 127 is the network that contains the IP loopback address. These special and reserved addresses are discussed later in this chapter.

Recall that Classes A, B, and C differ in where the dividing line is between the network ID and the host ID: 1 for network and 3 for host for Class A, 2 for each for Class B, and 3 for network and 1 for host for Class C. Based on this division, in Table 17-2, I have highlighted the network ID portion of the IP address ranges for each of Classes A, B, and C. The plain text corresponds to the range of host IDs for each allowable network ID. Figure 17-3 shows graphically how bits are used in each of the five classes.



**Figure 17-3: IP address class bit assignments and network/host ID sizes** This illustration shows how the 32 bits of IP address are assigned for each of the five IP address classes. Classes A, B, and C are the normal classes used for regular unicast addresses; each has a different dividing point between the network ID and host ID. Classes D and E are special and are not divided in this manner.

**KEY CONCEPT** In the classful IP addressing scheme, the class of an IP address is identified by looking at the first one, two, three, or four bits of the address. This can be done both by humans working with these addresses and routers making routing decisions. The use of these bit patterns means that IP addresses in different classes fall into particular address ranges that allow an address's class to be determined by looking at the first byte of its dotted decimal address.

For example, consider Class C. The lowest IP address is **192.0.0.0**, and the highest is **223.255.255.255**. The first three octets are the network ID and can range from **192.0.0** to **223.255.255**. For each network ID in that range, the host ID can range from 0 to 255.

**NOTE** It is common to see resources refer to the network ID of a classful address as including only the significant bits; that is, only the ones that are not common to all networks of that class. For example, you may see a Class B network ID shown in a diagram as having 14 bits, with the 10 that starts all such networks shown separately, as if it were not part of the network ID. Remember that the network ID does include those bits as well; it is 8 full bits for Class A, 16 for Class B, and 24 for Class C. In the case of Class D addresses, all 32 bits are part of the address, but only the lower 28 bits are part of the multicast group address; see the topic on multicast addressing later in this chapter for more.

## IP Address Class A, B, and C Network and Host Capacities

So far, I have introduced the concepts of IP address classes and showed how the classes relate to ranges of IP addresses. Of the five classes, D and E are dedicated to special purposes, so I will leave those alone for now. Classes A, B, and C are the ones actually assigned for normal (unicast) addressing purposes on IP internetworks, and therefore they are the primary focus of our continued attention.

As you've seen, the classes differ in the number of bits (and octets) used for the network ID compared to the host ID. The number of different networks possible in each class is a function of the number of bits assigned to the network ID, and likewise, the number of hosts possible in each network depends on the number of bits provided for the host ID. You must also take into account the fact that one, two, or three of the bits in the IP address are used to indicate the class itself, so it is effectively excluded from use in determining the number of networks (though again, it is still part of the network ID).

Based on this information, you can calculate the number of networks in each class, and for each class, the number of host IDs per network. Table 17-3 shows the calculations.

**Table 17-3:** IP Address Class Network and Host Capacities

IP Address Class	Total # of Bits for Network ID/Host ID	First Octet of IP Address	# of Network ID Bits Used To Identify Class	Usable # of Network ID Bits	Number of Possible Network IDs	# of Host IDs Per Network ID
Class A	8/24	0xxx xxxx	1	8-1 = 7	$2^7 - 2 = 126$	$2^{24} - 2 = 16,777,214$
Class B	16/16	10xx xxxx	2	16-2 = 14	$2^{14} = 16,384$	$2^{16} - 2 = 65,534$
Class C	24/8	110x xxxx	3	24-3 = 21	$2^{21} = 2,097,152$	$2^8 - 2 = 254$

Let's walk through one line of this table so you can see how it works using Class B as an example. The basic division is into 16 bits for network ID and 16 bits for host ID. However, the first 2 bits of all Class B addresses must be 10, so that leaves only 14 bits to uniquely identify the network ID. This gives us a total of  $2^{14}$  or 16,384 Class B network IDs. For each of these, you have  $2^{16}$  host IDs, less two, for a total of 65,534.

Why less two? For each network ID, two host IDs cannot be used: the host ID with all zeros and the ID with all ones. These are addresses with special meanings, as described in the next section. Also notice that two is subtracted from the number of network IDs for Class A. This is because two of the Class A network IDs (0 and 127) are reserved.

Several other address ranges are set aside in all three of the classes shown here. They are listed in the "IP Reserved, Private, and Loopback Addresses" section later in this chapter.

**KEY CONCEPT** In the classful IP addressing scheme, a Class A network contains addresses for about 16 million network interfaces; a Class B network contains about 65,000; and a Class C network contains 254.

As you can see, there is quite a disparity in the number of hosts available for each network in each of these classes. What happens if an organization needs 1,000 IP addresses? It must use either four Class Cs or one Class B (and in so doing, waste over 90 percent of the possible addresses in the Class B network). Bear in mind that there are only about 16,000 Class B network IDs available worldwide, and you begin to understand one of the big problems with classful addressing.

## IP Addresses with Special Meanings

Some IP addresses do not refer directly to specific hardware devices; instead, they are used to refer indirectly to one or more devices. To draw an analogy with language, most IP addresses refer to proper nouns, like “John” or “the red table in the corner.” However, some are used more the way you use pronouns such as “this one” or “that group over there.” I call these IP addresses with *special meanings*.

These special addresses are constructed by replacing the normal network ID or host ID (or both) in an IP address with one of two special patterns:

**All Zeros** When the network ID or host ID bits are replaced by a set of all zeros, the special meaning is the equivalent of the pronoun *this*, referring to whatever was replaced. It can also be interpreted as *the default* or *the current*. For example, if you replace the network ID with all zeros but leave the host ID alone, the resulting address means “the device with the host ID given, on *this network*,” or “the device with the host ID specified, on *the default network* or *the current network*.”

**All Ones** When the network ID or host ID bits are replaced by a set of all ones, this has the special meaning of *all*, meaning that the IP address refers to all hosts on the network. This is generally used as a broadcast address for sending a message to everyone.

**KEY CONCEPT** When the network ID or host ID of an IP address is replaced by a pattern of all ones or all zeros, the result is an address with a special meaning. Examples of such addresses include “all hosts” broadcast addresses and addresses that refer to a specific host or a whole network.

There are many special addresses. A small number apply to the entire TCP/IP network, while others exist for each network or host ID. Since two special patterns can be applied to the network ID, host ID, or both, there are six potential combinations, each of which has its own meaning. Of these, five are used.

Table 17-4 describes each of these special meanings and includes examples from Class A, B, and C. Note how an IP address in each of the common classes can be modified to have special meaning forms. (The first row shows the examples in their normal form, for reference.)

**Table 17-4:** IP Address Patterns with Special Meanings

Network ID	Host ID	Class A Example	Class B Example	Class C Example	Special Meaning and Description
Network ID	Host ID	77.91.215.5	154.3.99.6	227.82.157.160	<b>Normal Meaning:</b> Refers to a specific device.
Network ID	All Zeros	77.0.0.0	154.3.0.0	227.82.157.0	<b>The Specified Network:</b> This notation, with a 0 at the end of the address, refers to an entire network.
All Zeros	Host ID	0.91.215.5	0.0.99.6	0.0.0.160	<b>Specified Host on This Network:</b> This addresses a host on the current or default network when the network ID is not known or when it doesn't need to be explicitly stated.
All Zeros	All Zeros	0.0.0.0			<b>Me:</b> Used by a device to refer to itself when it doesn't know its own IP address. (Alternatively, "this host," or "the current/default host.") The most common use is when a device attempts to determine its address using a host-configuration protocol like DHCP. May also be used to indicate that any address of a multihomed host may be used.
Network ID	All Ones	77.255.255.255	154.3.255.255	227.82.157.255	<b>All Hosts on the Specified Network:</b> Used for broadcasting to all hosts on the local network.
All Ones	All Ones	255.255.255.255			<b>All Hosts on the Network:</b> Specifies a global broadcast to all hosts on the directly connected network. Note that there is no address that would imply sending to all hosts everywhere on the global Internet, since this would be very inefficient and costly.

**NOTE** The missing combination from Table 17-4 is that of the network ID being all ones and the host ID normal. Semantically, this would refer to "all hosts of a specific ID on all networks," which doesn't really mean anything useful in practice, so it's not used. Note also that, in theory, a special address where the network ID is all zeros and the host ID is all ones would have the same meaning as the all-ones limited broadcast address. The latter is used instead, however, because it is more general, not requiring knowledge of where the division is between the network ID and the host ID.

Since the all-zeros and all-ones patterns are reserved for these special meanings, they cannot be used for regular IP addresses. This is why, when you looked at the number of hosts per network in each of the classes, you had to subtract two from the theoretical maximum: one for the all-zeros case and one for the all-ones case.

Similarly, the network ID cannot be all zeros either. However, this doesn't require specific exclusion because the entire block of addresses with 0 in the first octet (0.x.x.x) is one of the reserved sets of IP addresses. These reserved addresses, described in the next section, further restrict the use of certain addresses in the IP address space for regular uses.

## **IP Reserved, Private, and Loopback Addresses**

In addition to the unusable numbers with special meanings just discussed, several other sets of IP addresses have special uses, and are therefore not available for normal address assignment. These generally fall into three categories: reserved, private, and loopback addresses.

### ***Reserved Addresses***

Several blocks of addresses were designated as reserved with no specific indication given as to what they were reserved for. Perhaps they were set aside for future experimentation or for internal use in managing the Internet. (In general, it's a good idea to set aside some portion of any limited resource for unanticipated needs.)

A couple of these blocks appear in each of the three main classes (A, B, and C), at the beginning and end of each class. (All of Class D and E are also reserved, since they aren't used for regular addressing.)

### ***Private, Unregistered, Nonroutable Addresses***

You'll recall that in the IP address overview in Chapter 16, I contrasted private and public IP addresses. Every IP address on an IP network must be unique. In the case of a public IP network, addresses are allocated by a central authority to ensure that there is no overlap. In contrast, on a private network, you can use whatever addresses you want.

Then why not just pick any random block of Class A, B, or C addresses for your private network and use that? You could, and some people did. For example, if you weren't connected to the Internet you could use, say, the Class A network 18.x.x.x that is reserved on the Internet to the Massachusetts Institute of Technology (MIT). Since you aren't connected to MIT, you would think that wouldn't matter.

However, as the Internet grew, those disconnected private networks needed to connect to the public Internet after all, and then they had a conflict. If they used the 18.x.x.x addresses, they would have to renumber all their devices to avoid getting a big bunch of computer geeks really angry. (There were, in fact, cases where companies that had used IP address space belonging to other companies accidentally connected those machines to the Internet, causing a small amount of ruckus in the process.)

RFC 1918 (superseding RFC 1597) provided the solution. It defines a set of unrouteable, special address blocks just for private addresses. These addresses simply don't exist on the public Internet. For this reason, they are not registered like other public addresses; they are sometimes called *unregistered*. Anyone can use them, but they cannot connect to the Internet because routers are not programmed to forward traffic with these address ranges outside of local organizations. RFC 1918 was published to encourage the use of these private blocks in order to cut down on the number of devices on the public Internet that didn't really need to be publicly accessible. This was in response to the need to conserve the public address space.

**NOTE** *In order to connect a network using private addressing to the public Internet, it is necessary to employ additional hardware and software. A gateway machine can be used as an interface between the public and private networks. Technologies such as Network Address Translation (NAT; see Chapter 28) are often used in conjunction with private IP addresses to allow these hosts to communicate on the public IP network.*

**KEY CONCEPT** Private address blocks were created to allow private IP Internets to be created using addresses that were guaranteed not to conflict with public IP addresses. They are commonly used in internetworks that aren't connected to the global Internet; devices using them can also access the global Internet by using NAT.

## Loopback Addresses

Normally, when a TCP/IP application wants to send information, that information travels down the protocol layers to IP, where it is encapsulated in an IP datagram. That datagram then passes down to the data link layer of the device's physical network for transmission to the next hop, on the way to the IP destination.

However, one special range of addresses, 127.0.0.0 to 127.255.255.255, is set aside for *loopback* functionality. IP datagrams sent by a host to a 127.x.x.x loopback address are not passed down to the data link layer for transmission; instead, they loop back to the source device at the IP level. In essence, this short-circuits the normal protocol stack; data is sent by a device's layer 3 IP implementation and then immediately received by it.

This loopback range is used for testing the TCP/IP protocol implementation on a host. Since the lower layers are short-circuited, sending to a loopback address allows you to isolate and test the higher layers (IP and above) without interference from the lower layers. 127.0.0.1 is the address most commonly used for testing purposes.

**KEY CONCEPT** Portions of the IP address space are set aside for reserved, private, and loopback addresses.

## **Reserved, Private, and Loopback Addressing Blocks**

Table 17-5 shows all of the special blocks set aside from the normal IP address space in numerical order, with a brief explanation of how each is used. It lists both the classful and the classless notation representing each of these blocks because the Internet now uses classless addressing, and because some of the private blocks don't correspond to single Class A, B, or C networks.

Note especially the private address block from 192.168.0.0 to 192.168.255.255. This is the size of a Class B network, but it isn't Class B in the classful scheme, because the first octet of 192 puts it in the Class C part of the address space. It is actually 256 contiguous Class C networks.

You may also notice the special Class B (/16) block 169.254.x.x. This is reserved for *Automatic Private IP Addressing (APIPA)*, discussed in Chapter 64. Systems that are configured to use this feature will automatically assign systems addresses from this block to enable them to communicate even if no server can be found for proper IP address assignment using the Dynamic Host Control Protocol (DHCP).

**Table 17-5:** Reserved, Private, and Loopback IP Addresses

<b>Range Start Address</b>	<b>Range End Address</b>	<b>Classful Address Equivalent</b>	<b>Classless Address Equivalent</b>	<b>Description</b>
<b>0.0.0.0</b>	<b>0.255.255.255</b>	Class A network 0.x.x.x	0/8	Reserved
<b>10.0.0.0</b>	<b>10.255.255.255</b>	Class A network 10.x.x.x	10/8	Class A private address block
<b>127.0.0.0</b>	<b>127.255.255.255</b>	Class A network 127.x.x.x	127/8	Loopback address block
<b>128.0.0.0</b>	<b>128.0.255.255</b>	Class B network 128.0.x.x	128.0/16	Reserved
<b>169.254.0.0</b>	<b>169.254.255.255</b>	Class B network 169.254.x.x	169.254/16	Class B private address block reserved for automatic private address allocation (see Chapter 64 for details)
<b>172.16.0.0</b>	<b>172.31.255.255</b>	16 contiguous Class B networks from 172.16.x.x through 172.31.x.x	172.16/12	Class B private address blocks
<b>191.255.0.0</b>	<b>191.255.255.255</b>	Class B network 191.255.x.x	191.255/16	Reserved
<b>192.0.0.0</b>	<b>192.0.0.255</b>	Class C network 192.0.0.x	192.0.0/24	Reserved
<b>192.168.0.0</b>	<b>192.168.255.255</b>	256 contiguous Class C networks from 192.168.0.x through 192.168.255.x	192.168/16	Class C private address blocks
<b>223.255.255.0</b>	<b>223.255.255.255</b>	Class C network 223.255.255.x	223.255.255/24	Reserved

## IP Multicast Addressing

The vast majority of traffic on IP internetworks is *unicast*, which is one source device sending to one destination device. IP also supports *multicasting*, which is a source device sending to a group of devices. Multicasting is not used a great deal on the present-day Internet, mainly due to a lack of widespread hardware support, though it is useful in certain circumstances, especially as a more efficient alternative to broadcasting.

The classful IP addressing scheme sets aside one-sixteenth of the address space for multicast addresses as Class D. Multicast addresses are identified by the pattern 1110 in the first four bits, which corresponds to a first octet of 224 to 239. Thus, the full range of multicast addresses is from 224.0.0.0 to 239.255.255.255.

Since multicast addresses represent a group of IP devices (sometimes called a *host group*), they can be used only as the destination of a datagram, never the source.

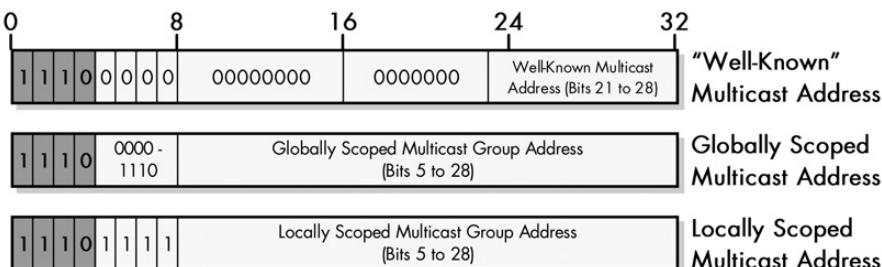
### Multicast Address Types and Ranges

The other 28 bits in the IP address define the *multicast group address*. The size of the Class D multicast address space is therefore  $2^{28}$ , or 268,435,456 multicast groups. No substructure defines the use of these 28 bits, and there is no specific concept of a network ID and host ID as in Class A, B, and C. However, certain portions of the address space are set aside for specific uses. Table 17-6 and Figure 17-4 show the general allocation of the Class D address space.

**Table 17-6:** IP Multicast Address Ranges and Uses

Range Start Address	Range End Address	Description
224.0.0.0	224.0.0.255	Reserved for special well-known multicast addresses
224.0.1.0	238.255.255.255	Globally scoped (Internetwide) multicast addresses.
239.0.0.0	239.255.255.255	Administratively scoped (local) multicast addresses

**NOTE** As with the other IP address classes, the entire 32 bits of the address is always used. It is only the least significant 28 bits that are interesting, because the upper four bits never change.



**Figure 17-4: IP Multicast address ranges and uses** All multicast addresses begin with 1110. The well-known group has zeros for the first 20 bits of the multicast group address, with 8 bits available to define 255 special multicast addresses. Multicast addresses starting with 1110 1111 are locally scoped; all other addresses are globally scoped (this includes addresses starting with 1110 0000 other than the 255 well-known addresses).

**RELATED INFORMATION** The concept of multicast address scope was more completely defined in IPv6, and I discuss it in more detail in the discussion of IPv6 multicast addresses in Chapter 25.

The bulk of the address space is in the middle multicast range. These are normal multicast addresses, like the Class A, B, and C unicast addresses, and they can be assigned to various groups.

The last address range is for *administratively scoped* multicast groups. This is a fancy term for multicast groups used within a private organization. This block, representing one-sixteenth of the total multicast address space, is comparable to the private addresses you saw earlier in this chapter. It is further subdivided into site-local multicast addresses, organization-local addresses, and so forth.

### **Well-Known Multicast Addresses**

The first block of 256 addresses is used to define special, well-known multicast address blocks (Table 17-7 has a selective listing). These do not represent arbitrary groups of devices and cannot be assigned in that manner. Instead, they have a special meaning that allows a source to send a message to a predefined group.

**Table 17-7:** Well-Known IP Multicast Addresses

<b>Range Start Address</b>	<b>Description</b>
224.0.0.0	Reserved; not used
224.0.0.1	All devices on the subnet
224.0.0.2	All routers on the subnet
224.0.0.3	Reserved
224.0.0.4	All routers using DVMRP
224.0.0.5	All routers using OSPF
224.0.0.6	Designated routers using OSPF
224.0.0.9	Designated routers using RIP-2
224.0.0.11	Mobile agents (for Mobile IP)
224.0.0.12	DHCP server/relay agent

Delivery of IP multicast traffic is more complex than unicast traffic due to the existence of multiple recipients. Instead of the normal resolution method through the Address Resolution Protocol (ARP) used for unicast datagrams, the IP multicast group and a hardware multicast group are mapped.

## **Problems with Classful IP Addressing**

The classful addressing system was the first major attempt to define a method for universal addressing of a large IP internetwork. There was a reasonable rationale for the system, as I mentioned in the overview of the classful scheme, and given that it was developed decades ago for a network that was limited in size, it did the job remarkably well for a long time.

No one ever expected the Internet to mushroom to anything close to its current size. As the Internet grew, the classful IP addressing mechanism showed some problems.

The three main problems with classful addressing are as follows:

**Lack of Internal Address Flexibility** Big organizations are assigned large, monolithic blocks of addresses that aren't a good match for the structure of their underlying internal networks.

**Inefficient Use of Address Space** The existence of only three block sizes (Classes A, B, and C) leads to a waste of limited IP address space.

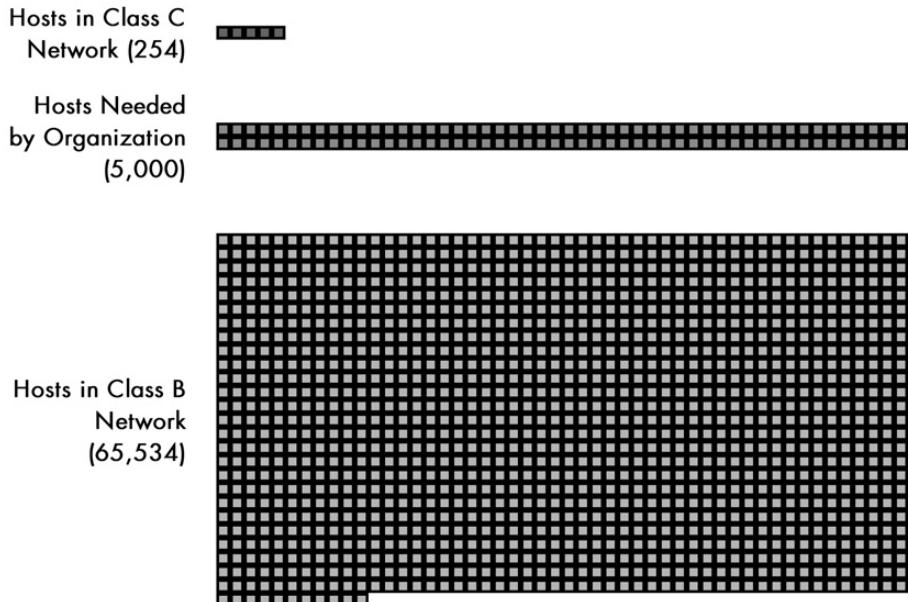
**Proliferation of Router Table Entries** As the Internet grows, more and more entries are required for routers to route IP datagrams. This causes performance problems for routers. Attempting to reduce inefficient address space allocation leads to even more router table entries.

The first issue results primarily from the fact that in the classful system, big companies are assigned a rather large (Class B) or truly enormous (Class A) block of addresses. They are considered by the Internet routers to be a single network, with one network ID. Now imagine that you are running a medium-to-large-sized company with 5,000 computers, and you are assigned a Class B address for your network. Do you really have 5,000 computers all hooked into a single network? I sure as heck hope you don't! Yet you would be forced to try to fit all of these into a single IP network in the original classful method. There was no way to create an internal hierarchy of addresses.

The second and third issues both stem from the fact that the granularity in the classful system is simply too low to be practical in a large internetwork; there are simply too few choices in the sizes of available networks. Three sizes seem fine in principle, but the gaps between the sizes are enormous, and the sizes don't match up well with the distribution of organizations in the real world. Consider the difference in size between Class C and Class B networks—a jump from 254 hosts all the way up to over 65,000! There are many, many companies that need more than 254 IP address but a lot fewer than 65,000. And what about Class A? How many companies need 16 *million* IP addresses, even the truly large ones? Probably none, if you think about it, yet that's half the IP address space right there.

What class of network should the company with 5,000 computers use? As Figure 17-5 shows, the classful scheme offers no good match for this company's needs. If it were assigned a Class B, over 90 percent of the IP addresses would be wasted.

The alternative to wasting all these IP addresses would be to give this fictitious company a bunch of Class C addresses instead of one Class B; but they would need 20 of them. While this would use the address space more efficiently, it leads to the third issue: Every router on the Internet then has to replace the single Class B router table entry with 20 Class C router entries. Multiply this by a few thousand medium-sized companies, and you can see that this method would add dramatically to the size of router tables. The larger these tables, the more time it takes for routers to make routing decisions.



**Figure 17-5: The main problem with classful addressing** In this scale diagram, each square represents 50 available addresses. Since a Class C address has only 254 addresses, and a Class B contains 65,534 addresses, an organization with 5,000 hosts is caught in the middle. It can only choose to either waste 90 percent of a Class B address or use 20 different Class C networks.

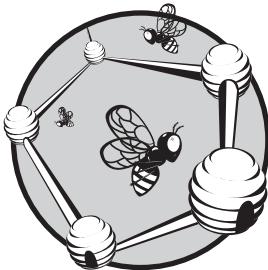
The problems with classful addressing have been solved by three enhancements, as you'll see in later chapters. The first, which primarily addresses the first issue, was the development of subnetting. The second was the move to classless addressing and routing, which replaces the classful system with a new method with higher granularity. This tackles the second and third issues by letting addresses be assigned based on real organizational needs, without requiring numerous routing table entries for each organization. The third improvement is the new IP version 6 (IPv6), which finally does away with the cramped 32-bit IP address space in favor of a gargantuan 128-bit one.

Other support technologies, such as NAT, have helped to extend the life of IPv4 by allowing multiple devices to share public addresses. This alone has added years to the life of the IPv4 addressing system.



# 18

## IP SUBNET ADDRESSING (SUBNETTING) CONCEPTS



In the previous chapter, we looked at the original classful IP addressing scheme, which conceptually divides a large inter-network into a simple two-level hierarchy that includes many *networks* of different sizes, each of which contains a number of *hosts*. The system works well for smaller organizations that may connect all their machines in a single network. However, it lacks flexibility for large organizations that often have many subnetworks, or *subnets*. To better meet the administrative and technical requirements of larger organizations, the classful IP addressing system was enhanced through a technique known as *subnet addressing*, or more simply, *subnetting*.

In this chapter, I describe the concepts and general techniques associated with IP subnet addressing. I begin with an overview of subnetting, including a discussion of the motivation for the system and its advantages. I discuss how the traditional two-level method for dividing IP addresses becomes three-level for subnetting. I talk about subnet masks and how they are used in calculations for addressing and routing. I discuss the default subnet masks used to represent the classful Class A, B, and C networks in a subnetting environment.

and then how custom subnet masks are used for these classes. I then discuss subnet identifiers and general concepts behind determining subnet and host addresses in a subnet environment. I provide summary tables for subnetting Class A, B, and C networks. I conclude with a brief discussion of *Variable Length Subnet Masking (VLSM)*, an enhancement of conventional subnetting that improves its flexibility further.

**NOTE** *I provide a great deal of coverage of subnetting, because understanding it is an important part of learning about how IP addresses work, and hence, how TCP/IP functions. However, the technique is today considered mostly historical because it is based on classful addressing. The concept of a subnet and subnet mask has certainly not disappeared, but the idea of being assigned a Class A, B, or C Internet address block and then explicitly subnetting it is no longer relevant.*

**RELATED INFORMATION** *This is the first of two chapters dedicated to IP address subnetting. Chapter 19 describes the step-by-step process for subnetting using examples. If you find that after reading this concepts section that you don't quite understand subnetting, try reading the example-based section, and you may find that it helps make it all click. On the other hand, if you are already somewhat familiar with subnetting, you may find that you can skip this concepts section and just go through the step-by-step examples. You will find much more in that chapter in the way of gory details of subnet mask, subnet address, and host address calculations. Putting the practical details there allows this section to concentrate on concepts without getting too bogged down in numbers.*

**BACKGROUND INFORMATION** *Understanding subnetting requires familiarity with binary numbers and how they are manipulated. This includes the concept of using boolean operators such as AND to "mask" binary digits. If reading that last sentence made you go "huh?" I strongly recommend reviewing the background section on computing mathematics (Chapter 4) before you proceed.*

## IP Subnet Addressing Overview, Motivation, and Advantages

As I discussed in the previous chapter, IP addressing was originally designed around the assumption of a strict two-level hierarchy for internetworks: the first level was the network, and the second level the host. Each organization was usually represented by a single network identifier (network ID) that indicated a Class A, B, or C block dedicated to them. Within that network, the organization needed to put all of the devices it wanted to connect to the public IP network.

It did not take long after this scheme was developed for serious inadequacies in it to be noticed, especially by larger organizations. In order to address this problem, RFC 950 [1985] defined a new addressing procedure called *subnet addressing* or *subnetting*.

Subnet addressing adds an additional hierarchical level to the way IP addresses are interpreted: Instead of having just hosts, the network has *subnets* and hosts. Each subnet is a subnetwork, and functions much the way a full network does in conventional classful addressing. A three-level hierarchy is thus created: networks, which contain subnets, each of which then has a number of hosts. Thus, an organization can organize hosts into subnets that reflect the way internal networks are structured. In essence, subnet addressing allows each organization to have its own internetwork within the Internet. This change brought numerous advantages over the old system, such as the following:

**Better Match to Physical Network Structure** Hosts can be grouped into subnets that reflect the way they are actually structured in the organization's physical network.

**Flexibility** The number of subnets and number of hosts per subnet can be customized for each organization. Each can decide on its own subnet structure and change it as required.

**Invisibility to Public Internet** Subnetting was implemented so that the internal division of a network into subnets is visible only within the organization. To the rest of the Internet, the organization is still just one big, flat network. This also means that any changes made to the internal structure are not visible outside the organization.

**No Need to Request New IP Addresses** Organizations don't need to constantly requisition more IP addresses, as they would in the workaround of using multiple small Class C blocks.

**No Routing Table Entry Proliferation** Since the subnet structure exists only within the organization, routers outside that organization know nothing about it. The organization still maintains a single (or perhaps a few) routing table entries for all of its devices. Only routers inside the organization need to worry about routing between subnets.

The change to subnetting affects both addressing and routing in IP networks. Addressing changes because, instead of having just a network ID and host ID, you now also have a *subnet ID* to be concerned with. The size of the subnet ID can vary for each network, so an additional piece of information is needed to supplement the IP address to indicate what part of the address is the subnet ID and what part is the host ID. This is a 32-bit number commonly called a *subnet mask*. The mask is used both for calculating subnet and host addresses, and by routers for determining how to move IP datagrams around a subnetted network.

Routing changes because of the additional level of hierarchy. In regular classful addressing, when a router receives an IP datagram, it only needs to decide if the destination is on the same network or a different network. Under subnetting, it must also look at the subnet ID of the destination and make one of three choices: same subnet, different subnet on the same network, or different network. Changes are also required to routing protocols, such as the Routing Information Protocol (RIP; see Chapter 38), to deal with subnets and subnet masks.

**KEY CONCEPT** Subnet addressing adds an additional hierarchical level to how IP addresses are interpreted by dividing an organization's IP network into subnets. This allows each organization to structure its address space to match its internal physical networks, rather than being forced to treat them a flat block. This solves a number of problems with the original classful addressing scheme, but requires changes to how addressing and routing work, as well as modifications to several TCP/IP protocols.

It's funny, but the main drawbacks to subnetting, compared with the older addressing scheme, have more to do with understanding how subnetting works than with the technology itself. More effort is required to deal with addressing and routing in a subnet environment, and administrators must learn how to subdivide

their network into subnets and properly assign addresses. This can be a bit confusing to someone who is new to subnetting. However, the technology today is quite well established, so even this is not much of a problem.

## IP Subnetting: Three-Level Hierarchical IP Subnet Addressing

As I mentioned earlier, subnetting adds an additional level to the hierarchy of structures used in IP addressing. To support this, IP addresses must be broken into three elements instead of two. This is done by leaving the network ID alone and dividing the host ID into a subnet ID and host ID. These subnet ID bits are used to identify each subnet within the network. Hosts are assigned to the subnets in whatever manner makes the most sense for that network.

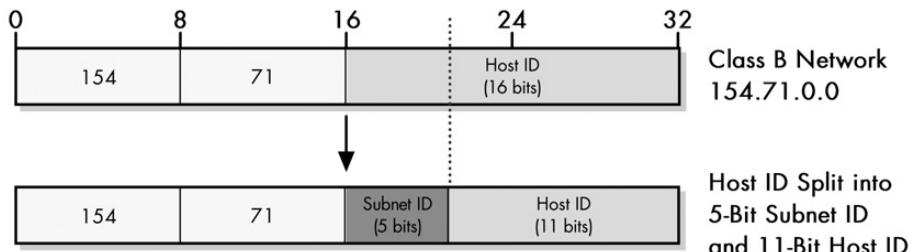
Interestingly, the earlier analogy to telephone numbers still holds in the world of subnetting and shows how subnetting changes the way IP addresses are interpreted. For example, a phone number like (401) 555-7777 has an area code (401) and a local number (555-7777). The local number, however, can itself be broken down into two parts: the exchange (555) and the local extension (7777). This means phone numbers really are comprised of three hierarchical components, just as IP addresses are in subnetting.

Of course, the number of bits in an IP address is fixed at 32. This means that in splitting the host ID into subnet ID and host ID, you reduce the size of the host ID portion of the address. In essence, you are stealing bits from the host ID to use for the subnet ID. Class A networks have 24 bits to split between the subnet ID and host ID; Class B networks have 16; and Class C networks have only 8.

**KEY CONCEPT** A classful network is subnetted by dividing its host ID portion, leaving some of the bits for the host ID while allocating others to a new subnet ID. These bits are then used to identify individual subnets within the network, into which hosts are assigned.

Now remember that when we looked at the sizes of each of the main classes in the previous chapter, we saw that, for each class, the number of networks and the number of hosts per network are a function of how many bits we use for each. The same applies to the splitting of the host ID. Since we are dealing with binary numbers, the number of subnets is two to the power of the size of the subnet ID field. Similarly, the number of hosts per subnet is two to the power of the size of the host ID field (less two for excluded special cases).

Let's take a brief example to see how this works. Imagine that you start with Class B network 154.71.0.0, with 16 bits for the network ID (154.71) and 16 are for the host ID. In regular classful addressing, there are no subnets and 65,534 hosts total. To subnet this network, you can decide to split those 16 bits however you feel best suits the needs of the network: 1 bit for the subnet ID and 15 for the host ID, or 2 and 14, 3 and 13, and so on. Most any combination will work, as long as the total is 16; I've used 5 and 11 in the example shown in Figure 18-1. The more bits you steal from the host ID for the subnet ID, the more subnets you can have, but the fewer hosts you can have for each subnet.



**Figure 18-1: Subnetting Class B network** We begin with the Class B network 154.71.0.0, which has 16 bits in its host ID block. We then subnet this network by dividing the host ID into a subnet ID and host ID. In this case, 5 bits have been allocated to the subnet ID, leaving 11 bits for the host ID.

Choosing how to split the host ID into subnet and host bits is one of the most important design considerations in setting up a subnetted IP network. The number of subnets is generally determined based on the number of physical subnetworks in the overall organizational network, and the number of hosts per subnetwork must not exceed the maximum allowed for the particular subnetting choice you make. Choosing how to divide the original host ID bits into subnet ID bits and host ID bits is sometimes called *custom subnetting* and is described in more detail later in this chapter.

## IP Subnet Masks, Notation, and Subnet Calculations

Subnetting divides an organization's network into a two-level structure of subnets and hosts that is entirely internal and hidden from all other organizations on the Internet. One of the many advantages of this is that each organization gets to make its own choice about how to divide the classful host ID into subnet ID and host ID.

In a nonsubnetted classful environment, routers use the first octet of the IP address to determine what the class of the address is, and from this they know which bits are the network ID and which are the host ID. When you use subnetting, these routers also need to know how that host ID is divided into subnet ID and host ID. However, this division can be arbitrary for each network. Furthermore, there is no way to tell how many bits belong to each simply by looking at the IP address.

In a subnetting environment, the additional information about which bits are for the subnet ID and which are for the host ID must be communicated to devices that interpret IP addresses. This information is given in the form of a 32-bit binary number called a *subnet mask*. The term *mask* comes from the binary mathematics concept called *bit masking*. This is a technique where a special pattern of ones and zeros can be used in combination with boolean functions such as AND and OR to select or clear certain bits in a number. (I explain bit masking in the background section on binary numbers and mathematics, in Chapter 4.)

### Function of the Subnet Mask

There's something about subnet masks that seems to set people's hair on end, especially if they aren't that familiar with binary numbers. However, the idea behind them is quite straightforward. The mask is a 32-bit number, just as the IP

address is a 32-bit number. Each of the 32 bits in the subnet mask corresponds to the bit in the IP address in the same location in the number. The bits of the mask in any given subnetted network are chosen so that the bits used for either the network ID or subnet ID are ones, while the bits used for the host ID are zeros.

**KEY CONCEPT** The *subnet mask* is a 32-bit binary number that accompanies an IP address. It is created so that it has a one bit for each corresponding bit of the IP address that is part of its network ID or subnet ID, and a zero for each bit of the IP address's host ID. The mask thus tells TCP/IP devices which bits in that IP address belong to the network ID and subnet ID, and which are part of the host ID.

Why bother doing this with a 32-bit binary number? The answer is the magic of boolean logic. You use the subnet mask by applying the boolean AND function between it and the IP address. For each of the 32 “bit pairs” in the IP address and subnet mask, you employ the AND function, the output of which is one only if both bits are one. What this means in practical terms is the following, for each of the 32 bits:

**Subnet Bit Is a One** In this case, you are ANDing either a zero or one in the IP address with a one. If the IP address bit is a zero, the result of the AND will be zero; if it is a one, the AND will be one. In other words, *where the subnet bit is a one, the IP address is preserved unchanged*.

**Subnet Bit Is a Zero** Here, you are ANDing with a zero, so the result is always zero, regardless of what the IP address is. Thus, *when the subnet bit is a zero, the IP address bit is always cleared to zero*.

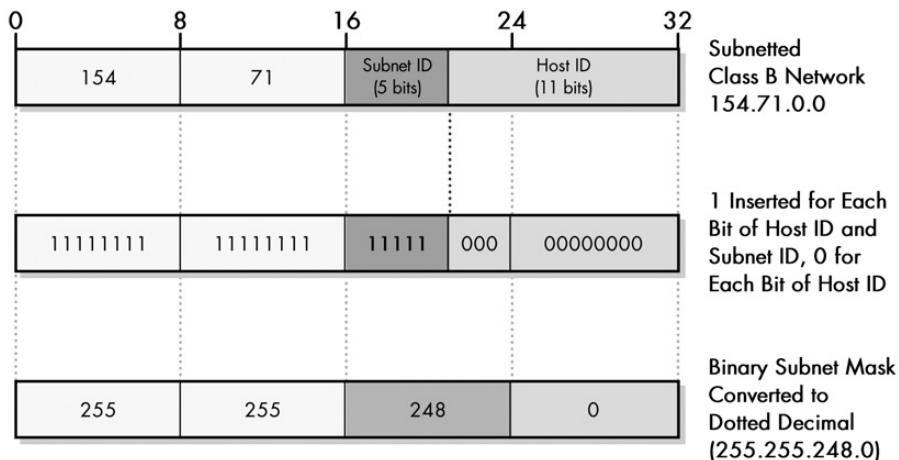
Thus, when you use the subnet mask on an IP address, the bits in the network ID and subnet ID are left intact, while the host ID bits are removed. Like a mask that blocks part of your face but lets other parts show, the subnet mask blocks some of the address bits (the host bits) and leaves others alone (the network and subnet bits). A router that performs this function is left with the address of the subnet. Since it knows from the class of the network what part is the network ID, it also knows what subnet the address is on.

**KEY CONCEPT** To use a subnet mask, a device performs a boolean AND operation between each bit of the subnet mask and each corresponding bit of an IP address. The resulting 32-bit number contains only the network ID and subnet ID of the address, with the host ID cleared to zero.

## ***Subnet Mask Notation***

Like IP addresses, subnet masks are always used as a 32-bit binary number by computers. And like IP addresses, using them as 32-bit binary numbers is difficult for humans. Therefore, they are usually converted to dotted decimal notation for convenience, just as IP addresses are.

For example, suppose you decide to subnet the Class B network 154.71.0.0 using 5 bits for the subnet ID and 11 bits for the host ID (see Figure 18-2). In this case, the subnet mask will have 16 ones for the network portion (since this is Class B) followed by 5 ones for the subnet ID, and 11 zeros for the host ID. That's 11111111 11111111 11111000 00000000 in binary, with the bits corresponding to the subnet ID highlighted. In dotted decimal, the subnet mask would be 255.255.248.0.



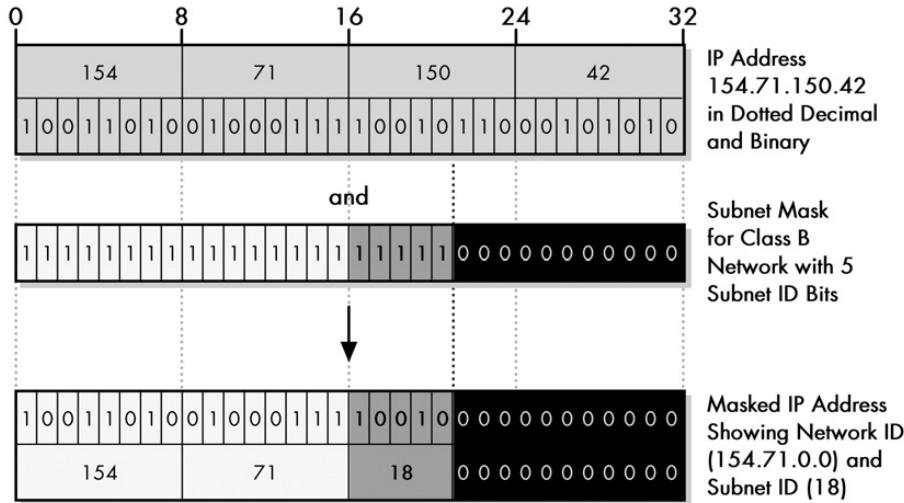
**Figure 18-2: Determining the subnet mask of a subnetted network** The Class B network from Figure 18-1 is shown at the top, with 5 bits assigned to the subnet ID and 11 bits left for the host ID. To create the subnet mask, you fill in a 32-bit number with 1 for each network ID and subnet ID bit, and 0 for each host ID bit. You can then convert this to dotted decimal.

### Applying the Subnet Mask: An Example

Now, let's see how the subnet mask might be used. Suppose you have a host on this network with an IP of 154.71.150.42 and a router needs to figure out which subnet this address is on. To do so, it performs the masking operation shown in Table 18-1 and Figure 18-3.

**Table 18-1: Determining the Subnet ID of an IP Address Through Subnet Masking**

Component	Octet 1	Octet 2	Octet 3	Octet 4
IP Address	10011010 (154)	01000111 (71)	10010110 (150)	00101010 (42)
Subnet Mask	11111111 (255)	11111111 (255)	11111000 (248)	00000000 (0)
Result of AND Masking	10011010 (154)	01000111(71)	10010000 (144)	00000000 (0)



**Figure 18-3: Determining the subnet ID of an IP address through subnet masking** Subnet masking involves performing a boolean AND between each corresponding bit in the subnet mask and the IP address. The subnet mask can be likened to a physical mask; each 1 in it lets the corresponding bit of the IP address show through, while each 0 blocks the corresponding IP address bit. In this way the host ID bits of the address are stripped so the device can determine the subnet to which the address belongs.

This result, 154.71.144.0, is the IP address of the subnet to which 154.71.150.42 belongs. There is no need to explicitly differentiate the network ID bits from the subnet ID bits, because you are still using classful addresses. Any router can see that since the first two bits of the address are 10, this is a Class B address. So the network ID is 16 bits, and this means the subnet ID must be bits 17 to 21, counting from the left. Here, the subnet is the portion highlighted earlier: 10010, or subnet 18. (I'll explain this better in the "IP Custom Subnet Masks" section later in this chapter.)

**KEY CONCEPT** The subnet mask is often expressed in dotted decimal notation for convenience, but is used by computers as a binary number and usually must be expressed in binary to understand how the mask works and the number of subnet ID bits it represents.

### Rationale for Subnet Mask Notation

In practical terms, the subnet mask actually conveys only a single piece of information: the line between the subnet ID and host ID. Then why bother with a big 32-bit binary number in that case, instead of just specifying the bit number where the division occurs? Instead of carrying the subnet mask of 255.255.248.0 around, why not just divide the IP address after bit 21? Even if devices want to perform a masking operation, couldn't they just create the mask as needed?

That's a very good question. There are two historical reasons: efficiency considerations and support for noncontiguous masks. The subnet mask expression is efficient because it allows routers to perform a quick masking operation to determine the subnet address. (This is not really an issue today given the speed of today's machines.)

When splitting the bits in the host ID for subnet ID and host ID, RFC 950 specifies that they may be split in more than one place. In the previous example, you could, instead of splitting the 16 bits into 5 bits for subnet ID and 11 for host ID, have done it as 2 bits for the subnet ID, then 4 bits for the host ID, then 3 more bits for the subnet ID, and finally 7 more bits for host ID. This would be represented by the subnet mask pattern 11000011 10000000 for those 16 bits (following the 16 ones for the network ID). Of course, subnetting this way makes assigning addresses *extremely* confusing. For this reason, while technically legal, noncontiguous subnet masking is not recommended and not done in practice.

Given that noncontiguous masks are not used, and today's computers are faster, the alternative method of expressing masks with just a single number is now often used. Instead of writing "IP address of 154.71.150.42 with subnet mask of 255.255.248.0," you can simply write "154.71.150.42/21." This is sometimes called *slash notation* or *Classless Inter-Domain Routing (CIDR) notation*. While this is more commonly used in Variable Length Subnet Masking (VLSM) environments and is the standard for specifying classless addresses under the CIDR addressing scheme (see Chapter 20), it is also sometimes seen in regular subnetting discussions.

**NOTE** Since these weird masks were never really used, some resources say that the subnet mask always had to be contiguous, but this is not true—originally, it was legal but advised against. Later, this practice became so out of favor that many hardware devices would not support it. Today, now that classless addressing and CIDR are standard, noncontiguous masks are simply illegal.

## IP Default Subnet Masks for Address Classes A, B, and C

In order to better understand how subnets divide a Class A, B, or C network, let's look at how the Class A, B, and C networks are represented in a subnetted environment. This might seem unnecessary if you aren't planning to create subnets, but the fact is, once subnetting became popular, most operating systems, networking hardware, and software assumed that subnetting would be used. Even if you decide not to subnet, you may need to express your unsubnetted network using a subnet mask.

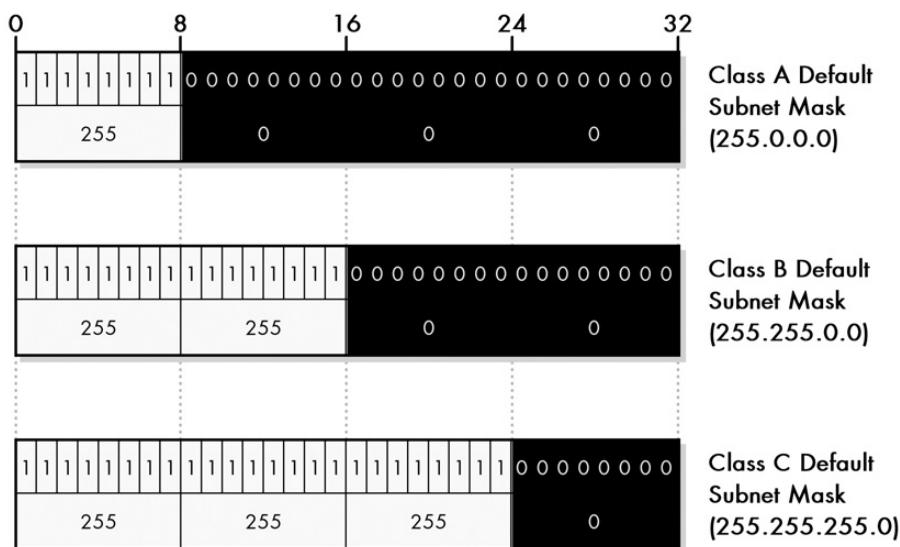
In essence, a nonsubnetted Class A, B, or C network can be considered the default for the more general, custom-subnetted network. You can think of a nonsubnetted network as being the case where you choose to divide the host ID so that exactly zero bits are used for the subnet ID, and all the bits are used for the host ID. This default case is the basis for the more practical subnetting you will examine shortly.

As is always the case, the subnet mask for a default, unsubnetted Class A, B, or C network has ones for each bit that is used for the network ID or subnet ID and zeros for the host ID bits. Of course, I just said you aren't subnetting, so there *are* no subnet ID bits! Thus, the subnet mask for this default case has ones for the network ID portion and zeros for the host ID portion. This is called the *default subnet mask* for each of the IP address classes.

Since Class A, B, and C divide the network ID from the host ID on octet boundaries, the subnet mask will always have all ones or all zeros in an octet. Therefore, the default subnet masks will always have 255s or 0s when expressed in decimal notation. Table 18-2 summarizes the default subnet masks for each of the classes. They are also shown graphically in Figure 18-4.

**Table 18-2:** Default Subnet Masks for Class A, Class B, and Class C Networks

IP Address Class	Total # of Bits for Network ID/Host ID	Default Subnet Mask			
		First Octet	Second Octet	Third Octet	Fourth Octet
Class A	8/24	11111111 (255)	00000000 (0)	00000000 (0)	00000000 (0)
Class B	16/16	11111111 (255)	11111111 (255)	00000000 (0)	00000000 (0)
Class C	24/8	11111111 (255)	11111111 (255)	11111111 (255)	00000000 (0)



**Figure 18-4:** Default subnet masks for Class A, Class B, and Class C networks

Thus, the three default subnet masks are 255.0.0.0 for Class A, 255.255.0.0 for Class B, and 255.255.255.0 for Class C.

While all default subnet masks use only 255 and 0, not all subnet masks with 255 and 0 are defaults. There are a small number of custom subnets that divide on octet boundaries as well. These are as follows:

**255.255.0.0** This is the default mask for Class B, but can also be the custom subnet mask for dividing a Class A network using 8 bits for the subnet ID (leaving 16 bits for the host ID).

**255.255.255.0** This is the default subnet mask for Class C, but can be a custom Class A with 16 bits for the subnet ID or a Class B with 8 bits for the subnet ID.

**KEY CONCEPT** Each of the three IP unicast and broadcast address classes, A, B, and C, has a *default subnet mask* defined that has a one for each bit of the class's network ID, a zero for each bit of its host ID, and no subnet ID bits. The three default subnet masks are 255.0.0.0 for Class A, 255.255.0.0 for Class B, and 255.255.255.0 for Class C.

## IP Custom Subnet Masks

A default subnet mask doesn't really represent subnetting because you are assigning zero bits to the subnet ID. To do real subnetting, you must dedicate at least one of the bits of the presubnetted host ID to the subnet ID.

Since you can choose the dividing point between subnet ID and host ID to suit the network, this is sometimes called *customized subnetting*. The subnet mask that you use when creating a customized subnet is, in turn, called a *custom subnet mask*. The custom subnet mask is used by network hardware to determine how you have decided to divide the subnet ID from the host ID in the network.

### Deciding How Many Subnet Bits to Use

The key decision in customized subnetting is how many bits to take from the host ID portion of the IP address to put into the subnet ID. You'll recall that the number of subnets possible on the network is two to the power of the number of bits you use to express the subnet ID, and the number of hosts possible per subnet is two to the power of the number of bits left in the host ID (less two, as I explain later in this section).

Thus, the decision of how many bits to use for each of the subnet ID and host ID represents a fundamental trade-off in subnet addressing:

- Each bit taken from the host ID for the subnet ID doubles the number of subnets that are possible in the network.
- Each bit taken from the host ID for the subnet ID (approximately) halves the number of hosts that are possible within each subnet on the network.

For example, say you start with a Class B network with the network address 154.71.0.0. Since this is Class B, 16 bits are for the network ID (154.71) and 16 are for the host ID. In the default case, there are no subnets and 65,534 hosts total. To subnet this network, you can use the following:

- One bit for the subnet ID and 15 bits for the host ID. If you do this, then the total number of subnets is  $2^1$ , or 2. The first subnet is 0, and the second is 1. The number of hosts available for each subnet is  $2^{15}-2$ , or 32,766.
- Two bits for the subnet ID and 14 for the host ID. In this case, you double the number of subnets. You now have  $2^2$ , or 4 subnets: 00, 01, 10, and 11 (subnets 0, 1, 2, and 3). But the number of hosts is now only  $2^{14}-2$ , or 16,382.
- Any combination of bits that add up to 16 as long as they allow you at least two hosts per subnet: 4 and 12, 5 and 11, and so on.

The way you decide to divide the classful host ID into subnet ID and host ID bits is the key design decision in subnetting. You make your choice based on the number of subnets in the network, and also on the maximum number of hosts that need to be assigned to each subnet in the network. For example, if you have 10 total subnets for your Class B network, you need 4 bits to represent this, because  $2^4$  is 16 while  $2^3$  is only 8. This leaves 12 bits for the host ID, for a maximum of 4,094 hosts per subnet.

However, suppose instead that you have 20 subnets. If so, 4 bits for subnet ID won't suffice; you need 5 bits ( $2^5=32$ ). This means that you now have only 11 bits for the host ID, for a maximum of 2,046 hosts per subnet. (Step 2 of the practical subnetting example in Chapter 19 discusses these decisions in more detail.)

Now if you have 20 subnets and also need a maximum of 3,000 hosts per subnet, you have a problem. You need 5 bits to express 20 different subnets, but you need 12 bits to express the number 3,000 for the host ID. That's 17 bits—too many. What's the solution? You might be able to shuffle your physical networks so that you only have 16. If not, you need a second Class B network.

**KEY CONCEPT** The fundamental trade-off in subnetting is that each addition of a bit to the subnet ID (and thus, subtraction of that bit from the host ID) doubles the number of subnets, and approximately halves the number of hosts in each subnet. Each subtraction of a bit from the subnet ID (and addition of that bit to the host ID) does the opposite.

## Determining the Custom Subnet Mask

Once you determine how many bits to devote to the subnet and host IDs, you can determine the subnet mask. You begin with the default subnet mask in binary for the appropriate class of the network. You start with the leftmost zero in that mask and change as many bits to one as you have dedicated to the subnet ID, at which point you can express the subnet mask in dotted decimal form. Figure 18-5 shows how the custom subnet mask can be determined for each of the subnetting options of a Class C network in both binary and decimal.

Consider the Class C network 200.13.94.0 in Figure 18-5. There are eight bits in the original host ID, which gives you six different subnetting options (you can't use seven or eight bits for the subnet ID, for reasons I will discuss shortly). Suppose you use three of these for the subnet ID, leaving five for the host ID. To determine the custom subnet mask, you start with the Class C default subnet mask:

11111111 11111111 11111111 00000000

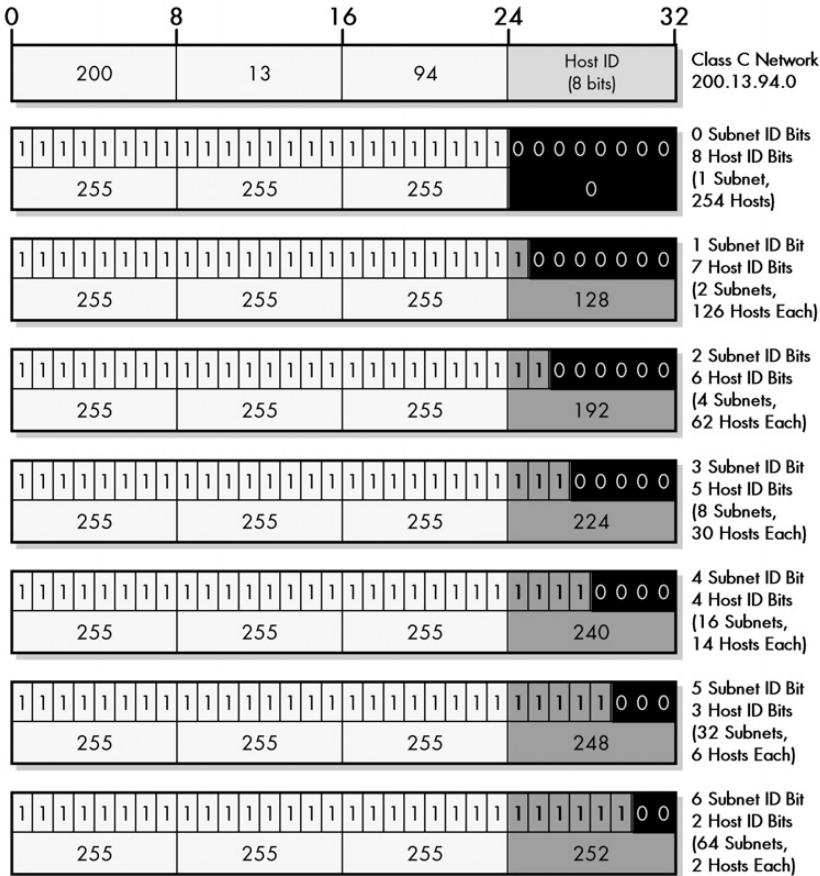
You then change the first three zeros to ones, to get the custom subnet mask:

11111111 11111111 11111111 11100000

In dotted decimal format, this is 255.255.255.224.

**NOTE** Once you've made the choice of how to subnet, you determine the custom subnet mask by starting with the default subnet mask for the network and changing each subnet ID bit from a zero to a one.

**NOTE** In regular subnetting, the choice of how many bits to use for the subnet ID is fixed for the entire network. You can't have subnets of different sizes—they must all be the same. Thus, the number of hosts in the largest subnet will dictate how many bits you need for the host ID. This means that in the previous case, if you had a strange configuration where 19 subnets had only 100 hosts each but the 20th had 3,000, you would have a problem. If this were the case, you could solve the problem easily by dividing that one oversized subnet into two or more smaller ones. An enhancement to subnetting called Variable Length Subnet Masking (VLSM) was created in large part to remove this restriction. VLSM is described later in the chapter.



**Figure 18-5: Custom subnet masks for Class C networks** Since there are host ID bits in a Class C network address, there are six different ways that the network can be subnetted. Each corresponds to a different custom subnet mask, which is created by changing the allocated subnet ID bits from zero to one.

### **Subtracting Two from the Number of Hosts per Subnet and (Possibly) Subnets per Network**

You've seen how you must subtract two from the number of hosts allowed in each network in regular classful addressing. This is necessary because two host IDs in each subnet have special meanings: the all-zeros host ID (for "this network") and the all-ones host ID (for broadcasts to all hosts on the network). These restrictions apply to each subnet under subnetting, too, which is why you must continue to subtract two from the number of hosts per subnet. (This is also why dividing the eight host ID bits of a Class C network into seven bits for subnet ID and one bit for host ID is meaningless: It leaves  $2^1 - 2 = 0$  hosts per subnet, which is not particularly useful.)

A similar issue occurs with the subnet ID as well. When subnetting was originally defined in RFC 950, the standard specifically excluded the use of the all-zeros and all-ones subnets. This was due to concern that routers might become confused by

these cases. A later standard, RFC 1812, “Requirements for IP Version 4 Routers,” removed this restriction in 1995. Thus, modern hardware now has no problem with the all-zeros or all-ones subnets, but some very old hardware may still balk at it.

**KEY CONCEPT** The number of hosts allowed in each subnet is the binary power of the number of host ID bits remaining after subnetting, less two. The reduction by two occurs because the all-zeros and all-ones host IDs within each subnet are reserved for two special meaning addresses: to refer to the subnetwork itself and to refer to its local broadcast address. In some implementations, the number of subnets is also reduced by two because the all-zeros and all-ones subnet IDs were originally not allowed to be used.

For this reason, you will sometimes see discussions of subnetting that exclude these cases. When that is done, you lose two potential subnets: the all-zeros and all-ones subnets. If you do this, then choosing one bit for subnet ID is no longer valid, as it yields  $2^1 - 2 = 0$  subnets. You must choose two bits if you need two subnets.

**NOTE** *In this book, I assume you are dealing with modern hardware and do not exclude the all-zeros and all-ones subnets, but I do try to make explicit note of this fact wherever relevant. Summary tables later in this chapter show the trade-off in subnetting each of Classes A, B, and C, and the subnet mask for each of the choices.*

## IP Subnet Identifiers, Subnet Addresses, and Host Addresses

The main advantage that conventional classful addressing without subnets offers over subnets is simplicity. For example, even though there can be problems with managing thousands of devices in a single Class B network, it is simple to assign addresses within the network: They are all lumped together, so any combination of bits can be used within the host ID (except for all-zeros and all-ones).

When you subnet, however, you create a two-level structure within the classful host ID: subnet ID and host ID. This means you must choose IP addresses for devices more carefully. In theory, you are selecting subnets to correspond to the physical networks within the organization, so you want to assign IP addresses in a way that is consistent with the physical network structure.

### ***Subnet Identifiers***

Once you decide how many subnets you will have, you need to identify the subnets and determine their addresses. You begin with the *subnet identifier*, the *subnet ID* of any subnets on our network. Subnets are numbered starting with zero and increasing up to one less than the maximum number of subnets, which is a function of how many bits are in the subnet ID. (If the all-zero and all-ones subnet IDs are excluded, as specified in RFC 950, then the first subnet ID is one.)

Of course, you may not need all of the subnets that can be defined. For example, if you have 20 subnets, you need five bits for the subnet identifier, which allows a theoretical maximum of 32 subnets. You would use only subnets 0 to 19; 20 through 31 would be reserved for future use. These subnets could be expressed either in decimal form (0, 1, 2 . . . up to 19) or in binary (00000, 00001, 00010, and so on, up to 10011).

## **Subnet Addresses**

For each subnet, you can also determine the *subnet address*. To do this, you start with the IP address for the overall network, which has all zeros in the classful host ID field (8 bits, 16 bits, or 24 bits). You then insert the subnet ID for a particular subnet into the designated subnet bits.

For example, to subnet the Class B network 154.71.0.0 shown in Figure 18-2, in which you use five subnet ID bits, you start with the following network IP address, with the subnet ID bits highlighted:

10011010 01000111 **00000000** 00000000

To find the address of say, subnet 11, you substitute 01011 for these bits, leaving the host ID bits zero, as follows:

10011010 01000111 **01011000** 00000000

You can then convert this from binary form to dotted decimal, resulting in a subnet address of 154.71.**88**.0.

**KEY CONCEPT** The *subnet identifier* of a subnet is just its subnet ID. The subnet address of a subnet is determined by substituting its subnet ID into the subnet bits of the overall network address.

When you look at subnet addressing, especially when you substitute subnet IDs in sequence, a pattern becomes immediately visible. The first subnet address is always the address of the overall network, because the subnet ID is all zeros. Then you find the second subnet address in decimal form by adding a specific multiple of two to one of the octets. The third address is then found by adding this same number to the second address, and so on.

In fact, the decimal value of each subnet address can be expressed as a formula, based on the class of the original network and the number of bits being used for the subnet ID. For example, consider a Class B network with the overall address of x.y.0.0 (it doesn't matter what x and y are for these purposes). Now say you are using two bits for the subnet ID. You have four subnet addresses here:

- The address of subnet 0 will be the same as the network address: x.y.0.0.
- The address of subnet 1 will be found by substituting 01 for the first two bits of the third octet. This yields an address of x.y.01000000.00000000, or x.y.64.0 in straight decimal.
- Subnet 2's address is found by substituting 10 for the subnet ID bits, so it is x.y.10000000.00000000, or x.y.128.0 in straight decimal.
- Subnet 3's address will be x.y.192.0.

So, the formula in this case for subnet  $N$  is  $x.y.N^2\cdot64.0$ . If you use five bits for a subnet, the formula is  $x.y.N^4\cdot8.0$ . As you saw earlier, the subnet address for subnet 11 in network 154.71.0.0 is 154.71.**88**.0. I have shown the formulas for all of the combinations of subnet ID and host ID size in the subnetting summary tables (Tables 18-3, 18-4, and 18-5). These formulas can be a real time-saver once you become more familiar with subnetting.

## **Host Addresses Within Each Subnet**

Once you know the subnet address for a particular subnet, you assign IP addresses by plugging in values into the remaining host ID bits. You skip the all-zeros value, so the first host in the subnet has all zeros for the host ID except for a one in the right-most bit position. Then the next host has all zeros except for “10” at the end (2 in decimal). You can do this all the way up to one less than the all-ones value. Again, you then convert each IP address from binary to decimal.

**NOTE** You can find exactly these details in Chapter 19’s coverage of practical subnetting.

## **IP Subnetting Summary Tables for Class A, Class B, and Class C Networks**

Since there are only a few options for how to subnet Class A, Class B, and Class C networks, I list the options for each class in summary Tables 18-3 through 18-5. These tables can help you quickly decide how many bits to use for subnet ID and host ID, and then what the subnet mask is for their selection. They also summarize nicely what I’ve discussed so far in this chapter.

Each row of each table shows one possible subnetting option for that class, including the number of bits for each of the subnet ID and host ID, and the number of subnets and hosts based on the number of bits. I then show the subnet mask in binary and decimal form, as well as in CIDR notation (covered in Chapter 20). Finally, I include the formula for calculating the addresses for each subnet under each of the options.

A few additional explanatory notes are in order regarding these tables:

- The values for the number of subnets per network assume that the all-zeros and all-ones subnets are allowed. If not, you must subtract two from those figures. This also means that the option using only one bit for the subnet ID becomes invalid, and the subnet address formulas no longer work as shown.
- The number of hosts per subnet excludes the all-zeros and all-ones cases, so it is two to the power of the number of host ID bits, less two.
- The first row of each table shows the default case where the number of subnet bits is zero, and thus the subnet mask is the default subnet mask for the class.
- In the subnet mask for all options but the default, I have highlighted the portion of the subnet mask corresponding to the subnet ID, for clarity. This has been done for each individual bit of the binary mask, and for each octet in the dotted decimal representation of the mask where part of the subnet ID is found.
- In looking at these tables, you will see that not all of the divisions make a great deal of sense in the real world, though you might be surprised. For example, at first glance, it seems silly to think that you might want to assign 14 bits of a Class B host ID to the subnet ID and leave 2 bits for the host ID—what sort of real network has 16,384 subnets with two hosts on each? Yet, some larger

Internet service companies may indeed require thousands of tiny subnets when setting up connections between routers or between their core network and their customers.

- The subnet address formulas in the last column of each table show the address for subnet  $N$  (numbering from zero up to one less than the maximum number of subnets). See the end of step 4 in the step-by-step subnetting discussion (Chapter 19) for a full explanation of how these formulas work.

**Table 18-3:** Subnetting Summary Table for Class A Networks

# of Subnet ID Bits	# of Host ID Bits	# of Subnets per Network	# of Hosts per Subnet	Subnet Mask (Binary/Dotted Decimal)	Subnet Mask (Slash/CIDR Notation)	Subnet Address #N Formula (N=0, 1, # of Subnets -1)
0 <b>(Default)</b>	<b>24</b>	1	16,277,214	11111111.00000000 .00000000.00000000 255.0.0.0	/8	—
1	<b>23</b>	2	8,388,606	11111111.10000000 .00000000.00000000 255. <b>128</b> .0.0	/9	x.N*128.0.0
2	<b>22</b>	4	4,194,302	11111111. <b>11000000</b> .00000000.00000000 255. <b>192</b> .0.0	/10	x.N*64.0.0
3	<b>21</b>	8	2,097,150	11111111. <b>11100000</b> .00000000.00000000 255. <b>224</b> .0.0	/11	x.N*32.0.0
4	<b>20</b>	16	1,048,574	11111111. <b>11110000</b> .00000000.00000000 255. <b>240</b> .0.0	/12	x.N*16.0.0
5	<b>19</b>	32	524,286	11111111. <b>11111000</b> .00000000.00000000 255. <b>248</b> .0.0	/13	x.N*8.0.0
6	<b>18</b>	64	262,142	11111111. <b>11111100</b> .00000000.00000000 255. <b>252</b> .0.0	/14	x.N*4.0.0
7	<b>17</b>	128	131,070	11111111. <b>11111110</b> .00000000.00000000 255. <b>254</b> .0.0	/15	x.N*2.0.0
8	<b>16</b>	256	65,534	11111111. <b>11111111</b> .00000000.00000000 255. <b>255</b> .0.0	/16	x.N.0.0
9	<b>15</b>	512	32,766	11111111. <b>11111111</b> .10000000.00000000 255. <b>255.128</b> .0	/17	x.N/2.(N%2)*128.0
10	<b>14</b>	1,024	16,382	11111111. <b>11111111</b> .11000000.00000000 255. <b>255.192</b> .0	/18	x.N/4.(N%4)*64.0

*(continued)*

**Table 18-3:** Subnetting Summary Table for Class A Networks (continued)

# of Subnet ID Bits	# of Host ID Bits	# of Subnets per Network	# of Hosts per Subnet	Subnet Mask (Binary/Dotted Decimal)	Subnet Mask (Slash/CIDR Notation)	Subnet Address #N Formula (N=0, 1, # of Subnets -1)
11	13	2,048	8,190	11111111.11111111 .11100000.00000000 255. <b>255.224.0</b>	/19	x.N/8.(N%8)*32.0
12	12	4,096	4,094	11111111.11111111 .11110000.00000000 255. <b>255.240.0</b>	/20	x.N/16.(N%16)*16.0
13	11	8,192	2,046	11111111.11111111 .11111000.00000000 255. <b>255.248.0</b>	/21	x.N/32.(N%32)*8.0
14	10	16,384	1,022	11111111.11111111 .11111100.00000000 255. <b>255.252.0</b>	/22	x.N/64.(N%64)*4.0
15	9	32,768	510	11111111.11111111 .11111110.00000000 255. <b>255.254.0</b>	/23	x.N/128.(N%128)*2.0
16	8	65,536	254	11111111.11111111 .11111111.00000000 255. <b>255.255.0</b>	/24	x.N/256.N%256.0
17	7	131,072	126	11111111.11111111 .11111111.10000000 255. <b>255.255.128</b>	/25	x.N/512. (N/2)%256.(N%2)*128
18	6	262,144	62	11111111.11111111 .11111111.11000000 255. <b>255.255.192</b>	/26	x.N/1024. (N/4)%256.(N%4)*64
19	5	524,288	30	11111111.11111111 .11111111.11100000 255. <b>255.255.224</b>	/27	x.N/2048. (N/8)%256.(N%8)*32
20	4	1,048,576	14	11111111.11111111 .11111111.11110000 255. <b>255.255.240</b>	/28	x.N/4096. (N/16)%256.(N%16)*16
21	3	2,097,152	6	11111111.11111111 .11111111.11111000 255. <b>255.255.248</b>	/29	x.N/8192. (N/32)%256.(N%32)*8
22	2	4,194,304	2	11111111.11111111 .11111111.11111100 255. <b>255.255.252</b>	/30	x.N/16384. (N/64)%256.(N%64)*4

**Table 18-4:** Subnetting Summary Table for Class B Networks

# of Subnet ID Bit	# of Host ID Bits	# of Subnets per Network	# of Hosts per Subnet	Subnet Mask (Binary/Dotted Decimal)	Subnet Mask (Slash/CIDR Notation)	Subnet Address #N Formula (N=0, 1, # of Subnets - 1)
0 (Default)	16	1	65,534	11111111.11111111 .00000000.00000000 255.255.0.0	/16	-
1	15	2	32,766	11111111.11111111 .10000000.00000000 255.255. <b>128.0</b>	/17	x.y.N*128.0
2	14	4	16,382	11111111.11111111. <b>11000000.00000000</b> 255.255. <b>192.0</b>	/18	x.y.N*64.0
3	13	8	8,190	11111111.11111111 .11100000.00000000 255.255. <b>224.0</b>	/19	x.y.N*32.0
4	12	16	4,094	11111111.11111111 .11110000.00000000 255.255. <b>240.0</b>	/20	x.y.N*16.0
5	11	32	2,046	11111111.11111111 .11111000.00000000 255.255. <b>248.0</b>	/21	x.y.N*8.0
6	10	64	1,022	11111111.11111111 .11111100.00000000 255.255. <b>252.0</b>	/22	x.y.N*4.0
7	9	128	510	11111111.11111111 .11111110.00000000 255.255. <b>254.0</b>	/23	x.y.N*2.0
8	8	256	254	11111111.11111111 .11111111.00000000 255.255. <b>255.0</b>	/24	x.y.N.0
9	7	512	126	11111111.11111111 .11111111.10000000 255.255. <b>255.128</b>	/25	x.y.N/2.(N%2)*128
10	6	1,024	62	11111111.11111111 .11111111.11000000 255.255. <b>255.192</b>	/26	x.y.N/4.(N%4)*64
11	5	2,048	30	11111111.11111111 .11111111.11100000 255.255. <b>255.224</b>	/27	x.x.N/8.(N%8)*32
12	4	4,096	14	11111111.11111111 .11111111.11110000 255.255. <b>255.240</b>	/28	x.y.N/16.(N%16)*16

*(continued)*

**Table 18-4:** Subnetting Summary Table for Class B Networks (continued)

# of Subnet ID Bit	# of Host ID Bits	# of Subnets per Network	# of Hosts per Subnet	Subnet Mask (Binary/Dotted Decimal)	Subnet Mask (Slash/CIDR Notation)	Subnet Address #N Formula (N=0, 1, # of Subnets -1)
13	3	8,192	6	11111111.11111111 .11111111.11111000 255.255.255.248	/29	x.y.N/32.(N%32)*8
14	2	16,384	2	11111111.11111111 .11111111.11111100 255.255.255.252	/30	x.y.N/64.(N%64)*4

**Table 18-5:** Subnetting Summary Table for Class C Networks

# of Subnet ID Bit	# of Host ID Bits	# of Subnets per Network	# of Hosts per Subnet	Subnet Mask (Binary/Dotted Decimal)	Subnet Mask (Slash/CIDR Notation)	Subnet Address #N Formula (N=0, 1, # of Subnets -1)
0 (Default)	8	1	254	11111111.11111111 .11111111.00000000 255.255.255.0	/24	—
1	7	2	126	11111111.11111111 .11111111.10000000 255.255.255.128	/25	x.y.z.N*128
2	6	4	62	11111111.11111111 .11111111.11000000 255.255.255.192	/26	x.y.z.N*64
3	5	8	30	11111111.11111111 .11111111.11100000 255.255.255.224	/27	x.y.z.N*32
4	4	16	14	11111111.11111111 .11111111.11110000 255.255.255.240	/28	x.y.z.N*16
5	3	32	6	11111111.11111111 .11111111.11111000 255.255.255.248	/29	x.y.z.N*8
6	2	64	2	11111111.11111111 .11111111.11111100 255.255.255.252	/30	x.y.z.N*4

## IP Variable Length Subnet Masking (VLSM)

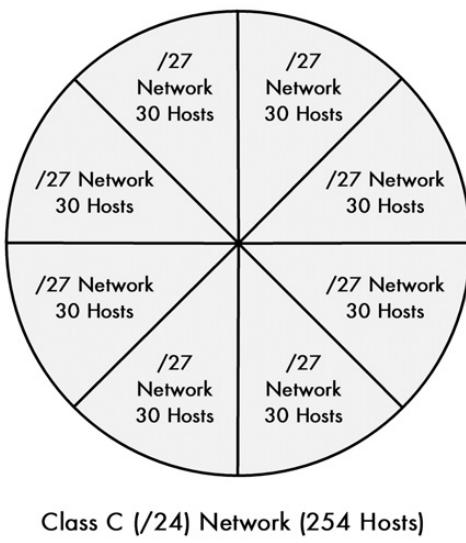
The main weakness with conventional subnetting is that the subnet ID represents only *one* additional hierarchical level in how IP addresses are interpreted and used for routing.

It may seem greedy to look at subnetting and say, “What, only *one* additional level?” However, in large networks, the need to divide the entire network into only one level of subnetworks doesn’t represent the best use of the IP address block.

Furthermore, you have already seen that since the subnet ID is the same length throughout the network, you can have problems if you have subnetworks with very different numbers of hosts on them. The subnet ID must be chosen based on whichever subnet has the greatest number of hosts, even if most of subnets have far fewer. This is inefficient even in small networks, and can result in the need to use extra addressing blocks while wasting many of the addresses in each block.

For example, consider a relatively small company with a Class C network, 201.45.222.0/24. The administrators have six subnetworks in their network. The first four subnets (S1, S2, S3, and S4) are relatively small, containing only 10 hosts each. However, one of them (S5) is for their production floor and has 50 hosts, and the last (S6) is their development and engineering group, which has 100 hosts. The total number of hosts needed is thus 190.

Without subnetting, the company has enough hosts in the Class C network to handle them all. However, when they try to subnet, they have a big problem. In order to have six subnets, they need to use three bits for the subnet ID. This leaves only five bits for the host ID, which means every subnet has the identical capacity of 30 hosts, as shown in Figure 18-6. This is enough for the smaller subnets but not enough for the larger ones. The only solution with conventional subnetting, other than shuffling the physical subnets, is to get another Class C block for the two big subnets and use the original for the four small ones. But this is expensive and means wasting hundreds of IP addresses!

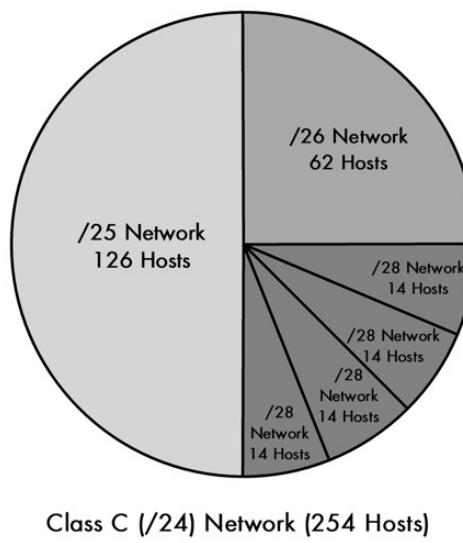


**Figure 18-6: Class C (/24) network split into eight conventional subnets** With traditional subnetting, all subnets must be the same size, which creates problems when there are some subnets that are much larger than others. Contrast this with Figure 18-7.

## The Solution: Variable Length Subnet Masking

The solution is an enhancement to the basic subnet addressing scheme called *Variable Length Subnet Masking* (VLSM). The idea is that you subnet the network and then subnet the subnets just the way you originally subnetted the network. In fact, you can do this multiple times, creating subnets of subnets of subnets, as many times as you need (subject to how many bits you have in the host ID of your address block).

It is possible to choose to apply this multiple-level splitting to only some of the subnets, thereby allowing you to selectively cut the IP address pie so that some of the slices are bigger than others. This means that the company in the previous example could create six subnets to match the needs of its networks, as shown in Figure 18-7.



**Figure 18-7: Class C (/24) network split using VLSM** Using VLSM, an organization can divide its IP network multiple times to create subnets that match the size requirements of its physical networks much better. Contrast this with Figure 18-6.

**KEY CONCEPT** *Variable Length Subnet Masking* (VLSM) is a technique for which subnetting is performed multiple times in iteration to allow a network to be divided into a hierarchy of subnetworks that vary in size. This allows an organization to better match the size of its subnets to the requirements of its networks.

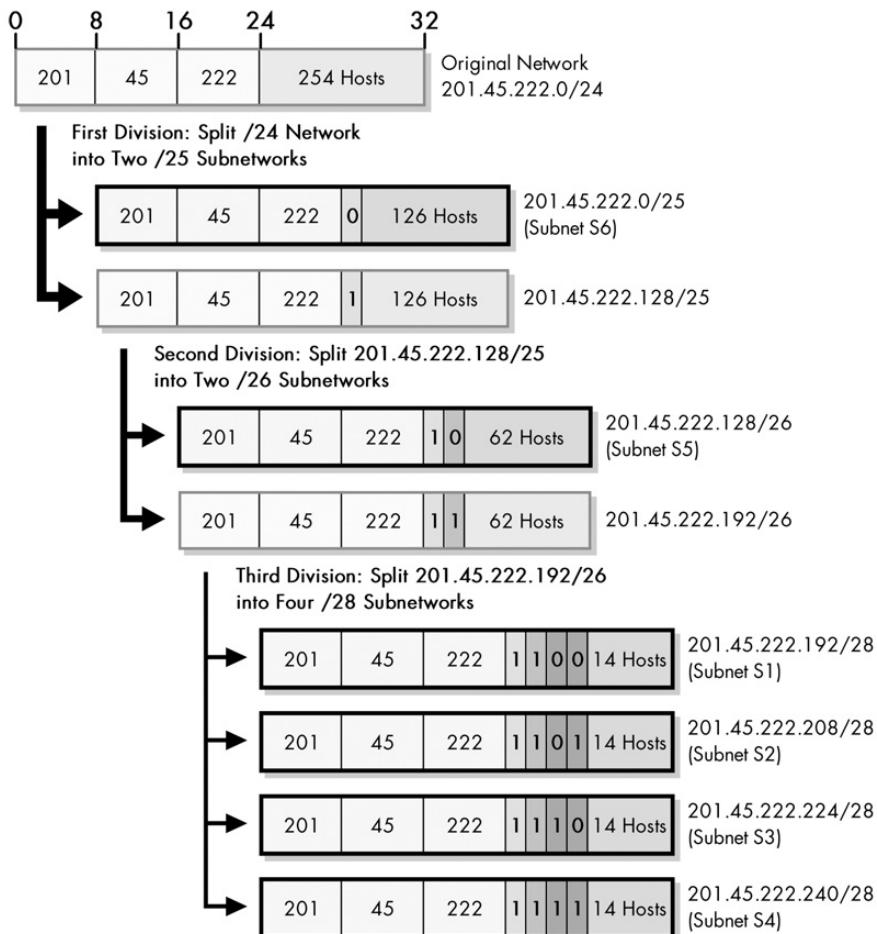
## Multiple-Level Subnetting Using VLSM

VLSM subnetting is done the same way as regular subnetting; it just involves extra levels of subnetting hierarchy. To implement it, you first subnet the network into large subnets and then further break down one or more of the subnets as required. You add bits to the subnet mask for each of the sub-subnets and sub-sub-subnets to reflect their smaller size.

In VLSM, the slash notation of classless addressing is commonly used instead of binary subnet masks (it works very much like CIDR), so that's what I will use.

**NOTE** If you're feeling a bit uncomfortable with how subnetting works, consider reading the chapter on practical subnetting (Chapter 19) before proceeding with the VLSM example that follows.

For example, consider the class C network, 201.45.222.0/24. You do three subnettings as follows (see Figure 18-8 for an illustration of the process).



**Figure 18-8: VLSM example** This diagram illustrates the example described in the text, of a Class C (/24) network divided using three hierarchical levels. It is first divided into two subnets; one subnet is divided into two sub-subnets; and one sub-subnet is divided into four sub-sub-subnets. The resulting six subnets, shown with thick black borders, have a maximum capacity of 126, 62, 14, 14, 14, and 14 hosts.

- You first do an initial subnetting by using one bit for the subnet ID, leaving you seven bits for the host ID and two subnets: 201.45.222.0/25 and 201.45.222.128/25. Each of these can have a maximum of 126 hosts. You set aside the first of these for subnet S6 and its 100 hosts.

- You take the second subnet, 201.45.222.128/25, and subnet it further into two sub-subnets by taking one bit from the seven bits left in the host ID. This gives you the sub-subnets 201.45.222.128/26 and 201.45.222.192/26, each of which can have 62 hosts. You set aside the first of these for subnet S5 and its 50 hosts.
- You take the second sub-subnet, 201.45.222.192/26, and subnet it further into four sub-sub-subnets. You take two bits from the six that are left in the host ID, which gives you four sub-sub-subnets that each can have a maximum of 14 hosts. These are used for S1, S2, S3, and S4.

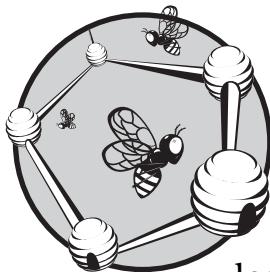
Although I've chosen these numbers so that they work out perfectly, you should get the picture. VLSM greatly improves both the flexibility and the efficiency of subnetting.

**NOTE** *In order to use VLSM, routers that support VLSM-capable routing protocols must be employed. VLSM also requires more care in how routing tables are constructed to ensure that there is no ambiguity in how to interpret an address in the network.*

As I mentioned earlier, VLSM is similar in concept to the way CIDR is performed. The difference between VLSM and CIDR is primarily one of focus. VLSM deals with subnets of a single network in a private organization. CIDR takes the concept you just saw in VLSM to the Internet as a whole by changing how organizational networks are allocated, replacing the single-level classful hierarchy with a multiple-layer hierarchy.

# 19

## IP SUBNETTING: PRACTICAL SUBNET DESIGN AND ADDRESS DETERMINATION EXAMPLE



When educators ask students what they consider to be the most confusing aspect in learning about networking, many say that it is IP address subnetting. While subnetting isn't all that difficult in concept, it can be a bit mind-boggling, in part due to the manipulations of binary numbers required. Many people understand the ideas behind subnetting but find it hard to follow the actual steps required to subnet a network.

For this reason, even though I explained the concepts behind subnetting in detail in the previous chapter, I felt it would be valuable to have another that provides a step-by-step look at how to perform custom subnetting. This chapter divides subnetting into five relatively straightforward stages that cover determining requirements; deciding how many bits to use for the subnet ID and host ID; and then determining important numbers such as the subnet mask, subnet addresses, and host addresses.

My focus here is on showing the practical "how" of subnetting. The topics work through two examples using a Class B and a Class C sample network to show you how subnetting is done, and I am explicit in showing

how everything is calculated. This means the section is a bit number heavy. Also, I try not to duplicate conceptual issues covered in the previous section, though a certain amount of overlap does occur. Overall, if you are not familiar with how subnetting works at all, you will want to read the previous chapter first. I do refer to topics in that chapter where appropriate, especially the summary tables. Incidentally, I only cover conventional subnetting here, not Variable Length Subnet Masking (VLSM).

This section may serve as a useful refresher or summary of subnetting for someone who is already familiar with the basics but just wants to review the steps performed in subnetting. Again, bear in mind that subnetting is based on the older, classful IP addressing scheme, and today's Internet is classless, using Classless Inter-Domain Routing (CIDR; see Chapter 20).

**NOTE** *If in reading this chapter, you find yourself wanting to do binary-to-decimal conversions or binary math, remember that most versions of Windows (and many other operating systems) have a calculator program that incorporates scientific functions.*

## IP Subnetting Step 1: Analyzing Requirements

When you are building or upgrading a network as a whole, the first step isn't buying hardware, or figuring out protocols, or even design. It's *requirements analysis*, the process of determining what it is the network needs to do. Without this foundation, you risk implementing a network that may perfectly match your design, but not meet the needs of your organization. The same rule applies to subnetting as well. Before you look at the gory details of host addresses and subnet masks, you must decide how to subnet the network. To do that, you must understand the requirements of the network.

Analyzing the requirements of the network for subnetting isn't difficult, because there are only a few issues that you need to consider. Since requirements analysis is usually done by asking questions, here's a list of the most important questions in analyzing subnetting requirements:

- What class is the IP address block?
- How many physical subnets are on the network today? (A *physical subnet* generally refers to a broadcast domain on a LAN—a set of hosts on a physical network bounded by routers.)
- Do you anticipate adding any more physical networks in the near future, and if so, how many?
- How many hosts do you have in the largest of the subnets today?
- How many hosts do you anticipate having in the largest subnet in the near future?

**KEY CONCEPT** To successfully subnet a network, you must begin by learning what the requirements of the network will be. The most important parameters to determine are the number of subnets required and the maximum number of hosts needed per subnet. Numbers should not be based on just present needs, but also take into account requirements anticipated in the near future.

The first question is important because everything in subnetting is based around dividing up a Class A, Class B, or Class C network, so you need to know which one you are dealing with. If you are in the process of designing a network from scratch and don't have a Class A, B, or C block yet, then you will determine which one you need based on the approximate size of the organization.

After that, you need to determine two key numbers: how many physical subnets you have and the maximum number of hosts per subnet. You need to know these not only for the present network, but for the *near future* as well. The current values for these two numbers represent how the network needs to be designed today. However, designing only for the present is not a good idea.

Suppose you have exactly four subnetworks in the network now. In theory, you could use only two bits for the subnet ID, since  $2^2$  equals 4. However, if the company were growing rapidly, this would be a poor choice. When you needed to add a fifth subnet, you would have a problem!

Similarly, consider the growth in the number of hosts in a subnet. If the current largest subnet has 60 hosts, you don't want six bits for the host ID, because that limits you to 62 hosts. You can divide large subnets into smaller ones, but this may just mean unnecessary additional work.

So what is the "near future?" The term is necessarily vague, because it depends on how far into the future the organization wants to look. On the one hand, planning for several years' growth can make sense, if you have enough IP addresses to do it. On the other, you don't want to plan too far out, since changes in the short term may cause you to completely redesign your network anyway.

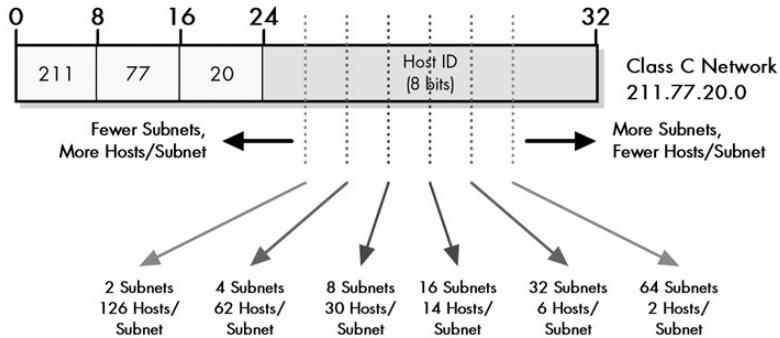
## IP Subnetting Step 2: Partitioning Network Address Host Bits

After you complete the brief requirements analysis, you should know the two critical parameters that you must have in order to subnet the network: the number of subnets required for the network and the maximum number of hosts per subnetwork. In using these figures to design the subnetted network, you will be faced with the key design decision in subnetting: how to divide the 8, 16, or 24 bits in the classful host ID into the subnet ID and host ID.

Put another way, you need to decide how many bits to steal from the host ID to use for the subnet ID. As I explained in the section on custom subnet masks in the previous chapter, the fundamental trade-off in choosing this number is as follows:

- Each bit taken from the host ID for the subnet ID doubles the number of subnets that are possible in the network.
- Each bit taken from the host ID for the subnet ID (approximately) halves the number of hosts that are possible within each subnet on the network.

There are six possible ways this decision can be made for a Class C network, as illustrated in Figure 19-1.



**Figure 19-1: Subnetting design trade-off for Class C networks** This drawing shows the options for subnetting a Class C network. As you increase the number of bits for the host ID, you increase the number of subnets, but decrease the size of each.

The relationship between the bits and the number of subnets and hosts is as follows:

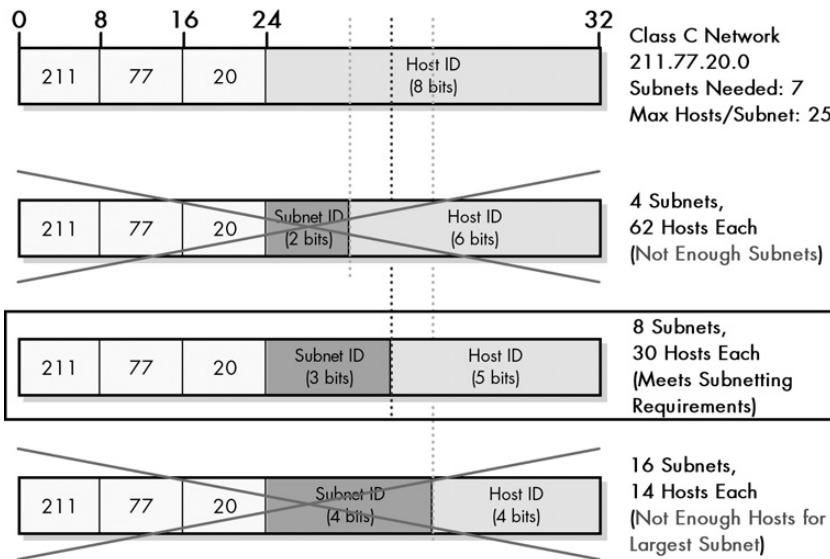
- The number of subnets allowed in the network is two to the power of the number of subnet ID bits.
- The number of hosts allowed per subnet is two to the power of the number of host ID bits, less two.

You subtract two from the number of hosts in each subnet to exclude the special meaning cases where the host ID is all zeros or all ones. As I explained in the previous chapter, this exclusion was originally also applied to the subnet ID, but is no longer in newer systems.

To choose how many bits to use for the subnet, you could use trial and error. By this, I mean you could try to first calculate the number of subnets and hosts when you use one bit for the subnet ID and leave the rest for the host ID. You could then try with two bits for the subnet ID, and then try with three, and so on. This would be silly, however; it's time-consuming and makes it hard for you to choose the best option. There's an easier method: You can use the subnetting summary tables, presented in the previous chapter. They let you look at all the options, and you can usually see immediately the best one for you.

### Class C Subnetting Design Example

Let's take an example. Suppose you have a Class C network, base address 211.77.20.0, with a total of seven subnets. The maximum number of hosts per subnet is 25. Looking at the subnetting summary table for Class C (Table 18-5 in Chapter 18), the answer is instantly clear: You need three bits for the subnet ID. Why? This allows you eight subnets and 30 hosts per subnet. If you try to choose two bits, you can't define enough subnets (only four). As Figure 19-2 shows, if you choose four bits for the subnet ID, then you can have only 14 hosts per subnet.



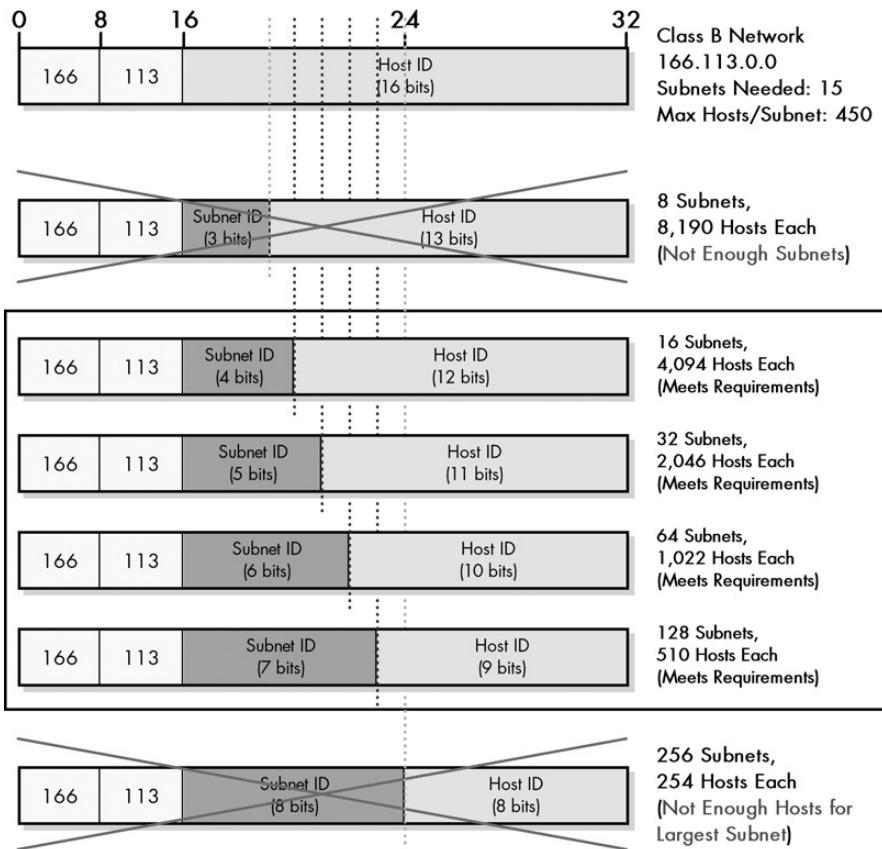
**Figure 19-2: Example of Class C subnetting** In this particular example, where seven subnets are needed and 25 hosts are needed for the largest subnet, there is only one choice of subnet ID size that meets the requirements. It's an easy decision!

### Class B Subnetting Design Example

In some cases, especially with larger networks, you may have multiple choices. Consider, as a more interesting example, the larger Class B network 166.113.0.0, where you have a total of 15 subnets and the largest has 450 hosts. Examining the subnet summary table for Class B (Table 18-4 in Chapter 18) suggests four acceptable options, as shown in Figure 19-3.

In all four of these options, the number of subnets is equal to 15 or greater, and the number of hosts per subnet is over 450. So which option should you choose? Usually, you want to pick something in the middle. If you use four bits for the subnet ID, this gives you a maximum of only 16 subnets, which limits growth in the number of subnets, since you already have 15. The same applies to the choice of seven bits for the subnet ID, since you already have 450 hosts in one subnet now, and that limits you to 510. Thus, you probably want either five or six bits here. If you expect more growth in the number of hosts in the largest subnet, you should choose five bits; if you expect more growth in the number of subnets, you should choose six bits. If you're unsure, it's probably best to assume more growth in the number of hosts per subnet, so here you would choose five bits.

The converse problem may also occur: You may be in a position where there don't appear to be any options—no rows in the summary table match. For example, if the Class C example had 35 hosts in the largest subnet instead of 25, you would be out of luck, because there is no combination of subnet ID and host ID size that works. The same is true in the Class B example if you had 4,500 hosts in that big subnet instead of 450. In this situation, you would need to divide the large subnet into a smaller one, use more than one IP address block, or upgrade to a larger block.



**Figure 19-3: Example of Class B subnetting** This Class B network needs at least 15 subnets and must allow up to 450 hosts per subnet. Three subnet ID bits are too few, and eight bits means only 254 hosts per subnet, which is insufficient. This leaves four acceptable options, so you must choose wisely.

**KEY CONCEPT** If there is more than one combination of subnet ID and host ID sizes that will meet requirements, try to choose a middle-of-the-road option that best anticipates future growth requirements. If no combination meets the requirements, the requirements have to change!

## IP Subnetting Step 3: Determining the Custom Subnet Mask

Once you have decided how many bits to use for the subnet ID and how many to leave for the host ID, you can determine the custom subnet mask for the network. Now, don't go running for cover on me. A lot of people's eyes glaze over at mention of the subnet mask, but it's really quite simple to figure out once you have done the homework in making the design decision you did in step 2. In fact, there are two ways of doing this; one is less work than the other, but they're both quite easy. I was going to call them the hard way and the easy way, but instead, I'll call them easy and easier.

## **Calculating the Custom Subnet Mask**

Let's start with the easy method, in which you calculate the subnet mask in binary form from the information you already have about the network, and then convert the mask to decimal. To refresh your memory and guide the process, remember this: The subnet mask is a 32-bit binary number where a one represents each bit that is part of the network ID or subnet ID, and a zero represents each bit of the host ID.

### **Class C Custom Subnet Mask Calculation Example**

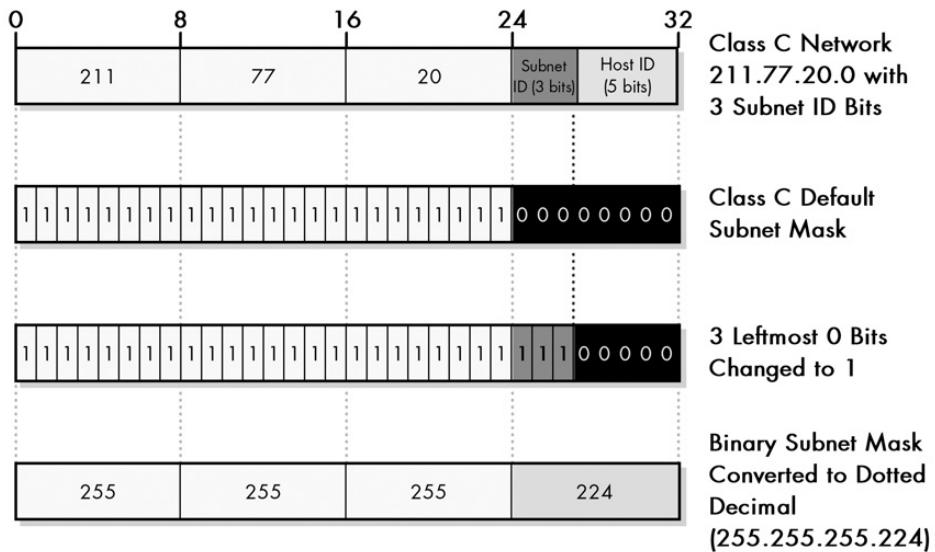
Refer back to the Class C example in the previous section (Figure 19-2). Say you decided to use three bits for the subnet ID, leaving five bits for the host ID. Here are the steps you will follow to determine the custom subnet mask for this network (illustrated in Figure 19-4):

1. **Determine Default Subnet Mask** Each of Classes A, B, and C has a default subnet mask, which is the subnet mask for the network prior to subnetting. It has a one for each network ID bit and a zero for each host ID bit. For Class C, the subnet mask is 255.255.255.0. In binary, this is:  
11111111 11111111 11111111 00000000
2. **Change Leftmost Zeros to Ones for Subnet Bits** You have decided to use three bits for the subnet ID. The subnet mask must have a one for each of the network ID or subnet ID bits. The network ID bits are already one from the default subnet mask, so, you change the three *leftmost* zero bits in the default subnet mask from a 0 to 1, as shown in bold here. This results in the following custom subnet mask for the network:  
11111111 11111111 11111111 **11100000**
3. **Convert Subnet Mask to Dotted Decimal Notation** You take each of the octets in the subnet mask and convert it to decimal. The result is the custom subnet mask in the form you usually see it: 255.255.255.224.
4. **Express Subnet Mask in Slash Notation** Alternatively, you can express the subnet mask in *slash notation*. This is just a slash followed by the number of ones in the subnet mask. 255.255.255.224 is equivalent to /27.

### **Class B Custom Subnet Mask Calculation Example**

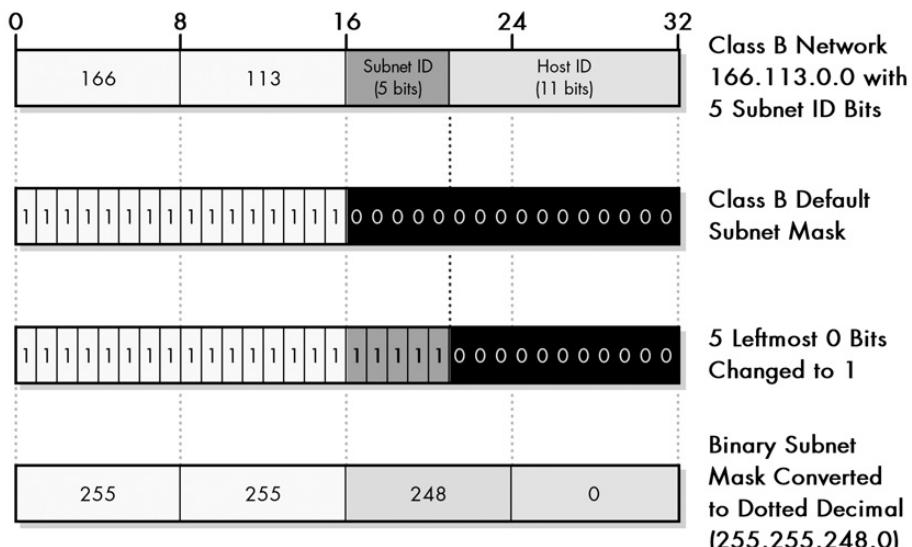
Now let's do the same example with the Class B network (166.113.0.0) with five bits for the subnet ID (with a bit less narration this time; see Figure 19-5):

1. **Determine Default Subnet Mask** For Class B, the subnet mask is 255.255.0.0. In binary, this is:  
11111111 11111111 00000000 00000000
2. **Change Leftmost Zeros to Ones for Subnet Bits** If you use five bits for the subnet ID, you change the five leftmost zero bits from a 0 to 1, as shown in bold, to give you the binary custom subnet mask, as follows:  
11111111 11111111 **11111000** 00000000



**Figure 19-4:** Determining the custom subnet mask for a Class C network

3. **Convert Subnet Mask to Dotted Decimal Notation** You take each of the octets in the subnet mask and convert it to decimal to give you a custom subnet mask of 255.255.248.0.
  4. **Express Subnet Mask in Slash Notation** You can express the subnet mask 255.255.248.0 as /21, since it is 21 ones followed by 11 zeros. In other words, its prefix length is 21.



**Figure 19-5:** Determining the custom subnet mask for a Class B network

## **Determining the Custom Subnet Mask Using Subnetting Tables**

Now, what could be easier than that? Well, you could simply refer to the subnetting summary tables, presented in Chapter 18. Find the table for the appropriate class, and then find the row that you selected in the previous step that matches the number of subnet ID bits you want to use. You can see the matching subnet mask right there.

(Hey, it's good to know how to do it yourself! You may not always have tables to refer to!)

## **IP Subnetting Step 4: Determining Subnet Identifiers and Subnet Addresses**

The network ID assigned to the network applies to the entire network. This includes all subnets and all hosts in all subnets. Each subnet, however, needs to be identified with a unique *subnet identifier*, or *subnet ID*, so it can be differentiated from the other subnets in the network. This is the purpose of the subnet ID bits that you took from the host ID bits in subnetting. After you have identified each subnet, you need to determine the address of each subnet, so you can use this in assigning hosts specific IP addresses.

This is another step in subnetting that is not really hard to understand or do. The key to understanding how to determine subnet IDs and subnet addresses is to always work in binary form, and then convert to decimal later. You will also look at a shortcut for determining addresses in decimal directly, which is faster but less conceptually simple.

**NOTE** I assume in this description that you will be using the all-zeros and all-ones subnet numbers. In the original RFC 950 subnetting system, those two subnets are not used, which changes most of the following calculations. See Chapter 18 for an explanation.

You number the subnets starting with 0, and then 1, 2, 3, and so on, up to the highest subnet ID that you need. You determine the subnet IDs and addresses as follows:

**Subnet ID** This is just the subnet number, and it can be expressed in either binary or decimal form.

**Subnet Address** This is the address formed by taking the address of the network as a whole and substituting the (binary) subnet ID for the subnet ID bits. You need to do this in binary, but only for the octets where there are subnet ID bits; the ones where there are only network ID bits or only host ID bits are left alone.

Seem complicated? Let's go back to the examples, and you'll see that it really isn't.

## **Class C Subnet ID and Address Determination Example**

You'll recall the Class C example network, 211.77.20.0. The network address in binary is as follows:

11010011 01001101 00010100 00000000

You are subnetting using three bits for the subnet ID, leaving five bits for the host ID. Now let's see the network address with the subnet bits in bold:

11010011 01001101 00010100 **00000000**

These are the bits that you substitute with the subnet ID for each subnet.

Notice that since the first three octets contain network ID bits, and the network ID is the same for every subnet, they never change. You don't even really need to look at them in binary form, though for clarity, you will do so here.

Here's how you determine the subnet IDs and addresses, again, starting with 0 (see Figure 19-6):

**Subnet 0** This has a subnet ID of 0, or 000 in binary. To find the address, you start with the network address in binary and substitute 000 for the subnet ID bits. Well gee, those bits are already all zero! What this means is that the address for subnet 0 is the same as the address for the network as a whole: 211.77.20.0. This is always the case: subnet 0 always has the same address as the network.

**Subnet 1** This has a subnet ID of 1 in decimal or 001 in binary. To find the address, you substitute 001 for the subnet ID bits, which yields the following:

11010011 01001101 00010100 **00100000**

Converting to decimal, you get 211.77.20.32.

**Subnet 2** This has a subnet ID of 2, or 010 in binary. To find its address, you substitute 010 for the subnet ID bits, to give you the following:

11010011 01001101 00010100 **01000000**

Which is 211.77.20.64 in binary.

**Subnet 3** This has a subnet ID of 011. As you can see, the first three octets of the address are always 211.77.20. The last octet here is **01100000**, which is 96 in decimal, so the whole address is 211.77.20.96.

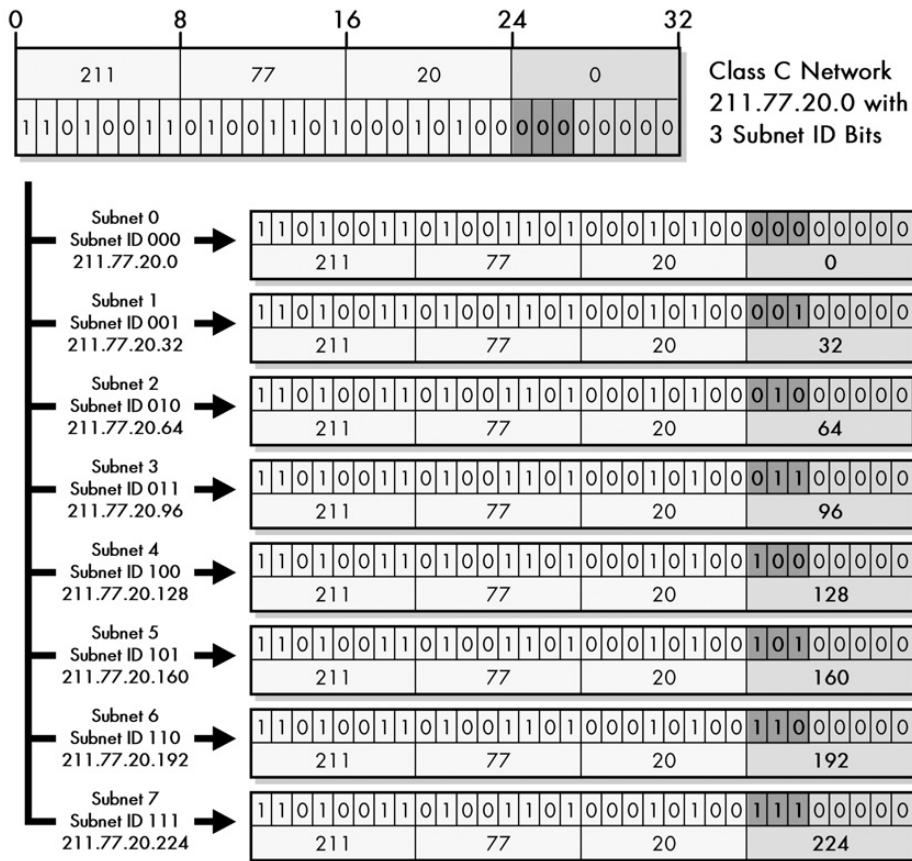
Starting to see a pattern here? Yes, the address of any subnet can be found by adding 32 to the last octet of the previous subnet. This pattern occurs for all subnetting choices; the increment depends on how many bits you are using for the subnet ID. Here, the increment is 32, which is  $2^5$ ; 5 is the number of host ID bits left after you took three subnet ID bits.

**Subnet 4** This has a subnet ID of 100. Its address is 211.77.20.128.

**Subnet 5** This has a subnet ID of 101. Its address is 211.77.20.160.

**Subnet 6** This has a subnet ID of 110. Its address is 211.77.20.192.

**Subnet 7** This has a subnet ID of 111. Its address is 211.77.20.224.



**Figure 19-6: Determining subnet addresses for a Class C network** This diagram shows each of the eight possible subnets created when you use three bits for the subnet ID in a Class C network. The binary subnet ID is simply substituted for the subnet bits, and the resulting 32-bit number is converted to dotted decimal form.

**KEY CONCEPT** The subnet addresses in a subnetted network are always evenly spaced numerically, with the spacing depending on the number of subnet ID bits.

This example needed only seven subnets, 0 through 6. Subnet 7 would be a spare. Notice that the last subnet has the same last octet as the subnet mask for the network? That's because I substituted 111 for the subnet ID bits, just as in the subnet mask calculation.

### Class B Subnet ID and Address Determination Example

Let's look at the other example now, Class B network 166.113.0.0. In binary this is as follows:

10100110 01110001 00000000 00000000

You're using five bits for the subnet ID, leaving 11 host ID bits. The network address with the subnet ID bits highlighted is as follows:

10100110 01110001 **00000000** 00000000

Here, only the third octet will ever change for the different subnets. The first two will always be 166.113, and the last octet will always be 0. There are 32 possible subnets; I'll list the first few so you can see the pattern (refer to Figure 19-7 as well):

**Subnet 0** This has a subnet ID of 00000. This means the address will be 166.113.0.0, which is the network address, as you would expect.

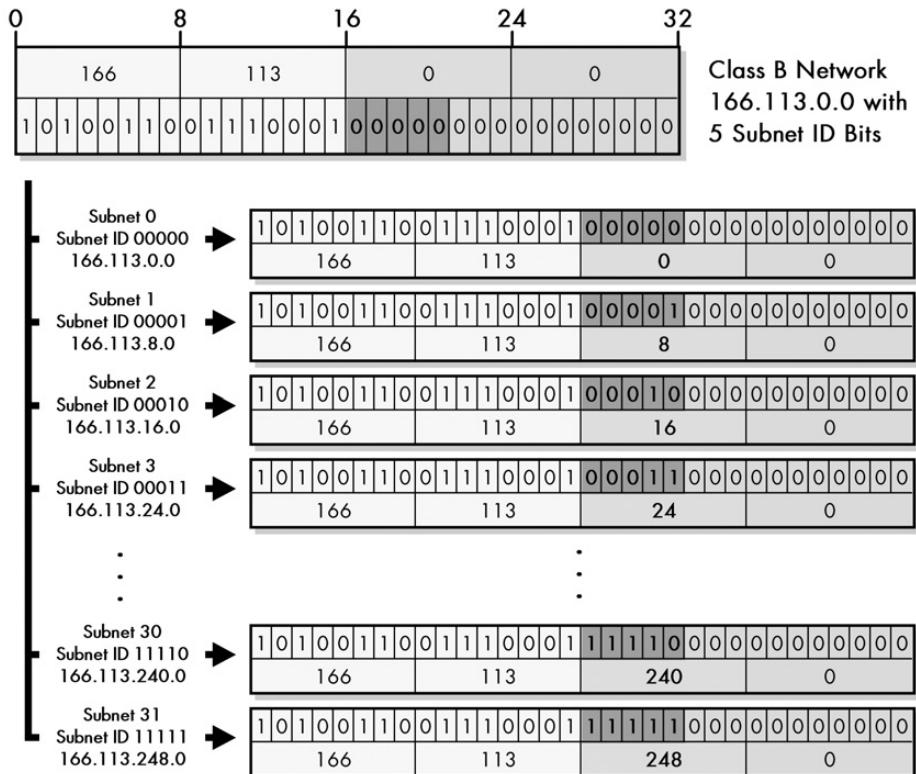
**Subnet 1** This has a subnet ID of 00001. The address becomes

10100110 01110001 **00001000** 00000000

This is 116.113.8.0 in decimal.

**Subnet 2** This has a subnet ID of 00010, giving an address of 116.113.**00010000**.0 or 116.113.16.0.

**Subnet 3** This has a subnet ID of 00011 and a subnet address of 116.113.24.0.



**Figure 19-7: Determining subnet addresses for a Class B network** This is the same as Figure 19-6, but for a Class B network with five subnet ID bits (I have not shown all 32 subnets, for obvious reasons).

Again, the pattern here is obvious: You add eight to the third octet to get successive addresses. The last subnet here is 31, which has a subnet address of 166.113.248.0, which has the same third and fourth octets as the subnet mask of 255.255.248.0.

## **Using Subnet Address Formulas to Calculate Subnet Addresses**

Since the subnet addresses form a pattern, and the pattern depends on the number of subnet ID bits, it is possible to express the subnet addresses using a single formula for each subnetting option. I have shown these formulas for each of the Classes A, B, and C in the subnetting summary tables in Chapter 18. The formulas can be used to directly calculate the address of subnet  $N$ , where  $N$  is numbered from 0 up to one less than the total number of subnets, as I have done earlier.

In these formulas, the network ID bits are shown as x., x.y., or x.y.z. for the three classes. This just means that the subnet addresses have as those octets whatever the numbers are in those octets for the network address. In the examples, x.y would be 166.113 for the Class B network, and x.y.z would be 211.77.20 for the Class C network.

When the number of subnet bits is eight or less, the formula is relatively simple, and a calculation is done for only one octet, as a multiplication of  $N$ , such as  $N^*4$  or  $N^*32$ . This is usually the case, since the number of subnets is usually less than 256, and it's the case with both of the examples.

In the Class C network with three subnet ID bits, the formula from the table is x.y.z. $N^*32$ . For this network, all subnets are of the form 211.77.20. $N^*32$ , with  $N$  going from zero to seven. So, subnet 5 is 211.77.20.(5 $^*32$ ), which is 211.77.20.160, as you saw before. Similarly, in the Class B network with five subnet ID bits, the formula is x.y. $N^*8.0$ . In this case, x.y is 166.113. Subnet 26 would have the address 166.113.(26 $^*8$ ).0, or 166.113.208.0.

This is pretty simple stuff, and it makes the formulas a good shortcut for quickly determining subnet addresses, especially when there are many subnets. They can also be used in a spreadsheet.

The only place where using the formulas requires a bit of care is when the number of subnet bits is nine or more. This means that the subnet identifier crosses an octet boundary, and this causes the formula to become more complex.

When the number of subnet bits is greater than eight, some of the octets are of the form  $N$  divided by an integer, such as  $N/8$ . This is an integer division, which means divide  $N$  by 8, keep the integer part, and drop the fractional part or remainder. Other octets are calculated based on the modulo of  $N$ , shown as  $N\%8$ . This is the exact opposite: It means divide  $N$  by 8, drop the integer, and keep the remainder. For example, 33/5 in integer math is 6 (6 with a remainder of 3, drop the remainder, or alternately, 6.6, drop the fraction), and 33%5 is 3 (6 with a remainder of 3, drop the 6, keep the remainder).

Let's take as an example the Class B network and suppose that for some strange reason you decided to use ten bits for the subnet ID instead of five. In this case, the formula is x.y. $N/4.(N\%4)^*64$ . Subnet 23 in this case would have the address 166.113.23/4.(23%4) $^*64$ . The 23/4 becomes just 5 (the fractional .75 is dropped). 23 modulo 4 is 3, which is multiplied by 64 to get 192. So the subnet address is 166.113.5.192. Subnet 709 would be 116.113.709/4.(709%4) $^*64$ , which is 116.113.177.64.

Okay, now for the real fun! If you subnet a Class A address using more than 16 bits for the subnet ID, you are crossing *two* octet boundaries, and the formulas become very . . . interesting, involving both integer division *and* modulo. Suppose you were in charge of Class A address 21.0.0.0 and decide to subnet it. However, you sat down to do this after having had a few stiff ones at the office holiday party, so your judgment is a bit impaired. You decide that it would be a great idea to choose 21 bits for the subnet ID, since you like the number 21. This gives you a couple million subnets.

The formula for subnet addresses in this case is rather long and complicated:  $x.N/8192.(N/32)\%256.(N\%32)*8$ . Yikes. Well, this is a bit involved—so much so that it might be easier to just take a subnet number and do it in binary, the long way. But let's take an example and see how it works for, say, subnet 987654. The first octet is 21. The second octet is 987654/8192, integer division. This is 120. The third octet is  $(987654/32)\%256$ . The result of the division is 30864 (you drop the fraction). Then you take  $30864\%256$ , which yields a remainder of 144. The fourth octet is  $(987654\%32)*8$ . This is  $6*8$  or 48. So subnet address 987654 is 21.120.144.48.

(Don't drink and drive. Don't drink and subnet either.)

## IP Subnetting Step 5: Determining Host Addresses for Each Subnet

Once you know the addresses of each of the subnets in the network, you use these addresses as the basis for assigning IP addresses to the individual hosts in each subnet. You start by associating a subnet base address with each physical network (since at least in theory, the subnets correspond to the physical networks). You then sequentially assign hosts particular IP addresses within the subnet (or in a different manner, if you prefer!).

Determining host addresses is really quite simple once you know the subnet address. All you do is substitute the numbers 1, 2, 3, and so on for the host ID bits in the subnet address. You must do this in binary and then convert the address to decimal form. Again, you can take some shortcuts once the rather obvious pattern of how to assign addresses emerges. You'll look at those near the end of the chapter.

### Class C Host Address Determination Example

Let's start with the Class C example again, 211.77.20.0, which you divided into eight subnets using three subnet bits. Here's how the address appears with the subnet bits shown in bold, and the host ID bits shown in italics:

11010011 01001101 00010100 **00000000**

The first subnet is subnet 0, which has all zeros for those subnet bits, and thus the same address as the network as a whole: 211.77.20.0. You substitute the numbers 1, 2, 3, and so on for the italicized bits to get the host IDs. (Remember that you don't start with zero here because for the host ID, the all-zeros and all-ones binary patterns have special meaning.) So it goes like this:

The first host address has the number 1 for the host ID, or 00001 in binary. So it is as follows:

**11010011 01001101 00010100 **00000001****

In decimal, this is 211.77.20.1.

The second host address has the number 2 for the host ID, or 00010 in binary. Its binary value is as follows:

**11010011 01001101 00010100 **00000010****

In decimal, this is 211.77.20.2.

I'm sure you get the picture already; the third host will be 211.77.20.3, the fourth 211.77.20.4, and so on. There is a maximum of 30 hosts in each subnet, as you saw earlier. So the last host in this subnet will be found by substituting 30 (11110 in binary) for the host ID bits, resulting in a decimal address of 211.77.20.30.

You can do the same thing for each of the other subnets; the only thing that changes is the values in the subnet ID bits. Let's take subnet 6, for example. It has 110 for the subnet bits instead of 000. So its subnet base address is 211.77.20.192, or

**11010011 01001101 00010100 **11000000****

You assign hosts to this subnet by substituting 00001, then 00010, then 00011 for the host ID bits as shown earlier. Let's take the hosts one at a time:

The first host address is as follows:

**11010011 01001101 00010100 **11000001****

or 211.77.20.193.

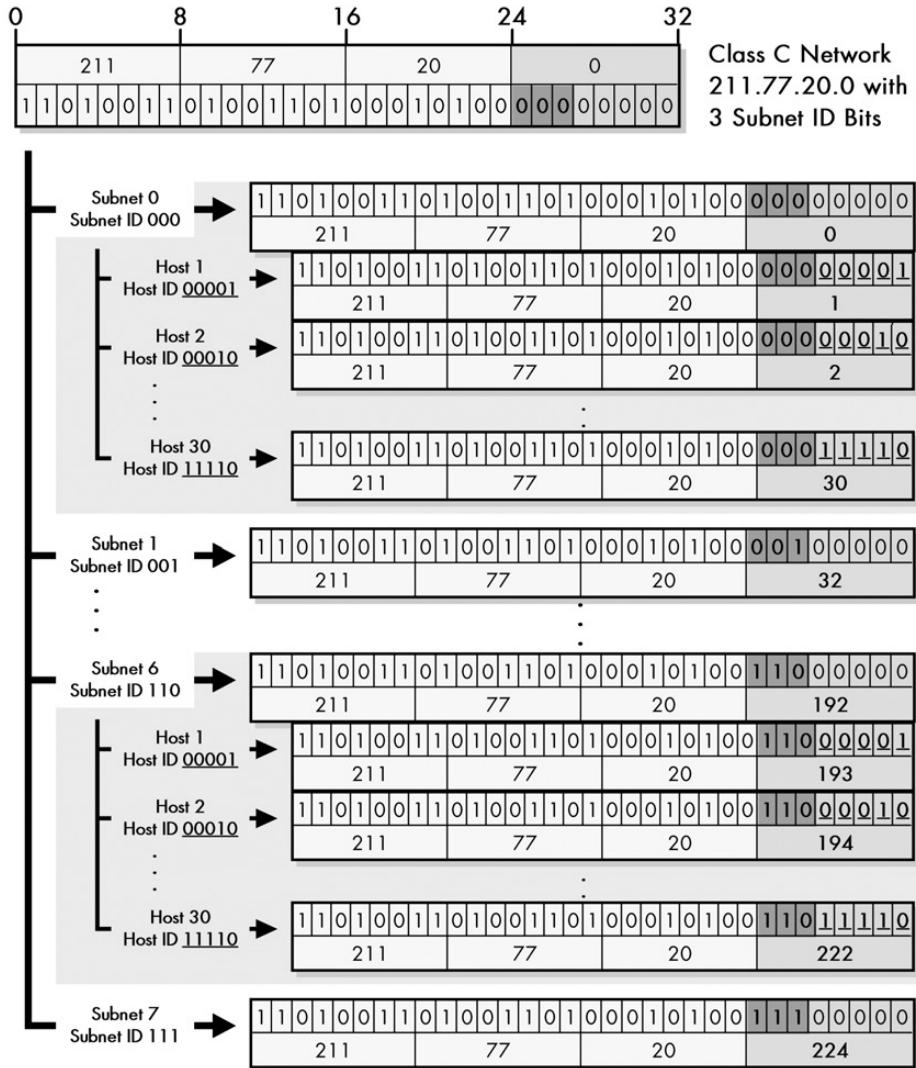
The second host address is

**11010011 01001101 00010100 **11000010****

or 211.77.20.194.

And so on, all the way up to the last host in the subnet, which is 211.77.20.222. Figure 19-8 shows graphically how subnet and host addresses are calculated for this sample network.

One more address you may wish to calculate is the broadcast address for the subnet. This is one of the special cases, as discussed in Chapter 18, found by substituting all ones for the host ID. For subnet 0, this would be 211.77.20.31. For subnet 6, it would be 211.77.20.223. That's pretty much all there is to it.



**Figure 19-8: Determining host addresses for a Class C network** This diagram shows how both subnet addresses and host addresses are determined in a two-step process. The subnet addresses are found by substituting subnet ID values (shown in bold) for the subnet ID bits of the network. Then, for any given subnet address, you can determine a host address by substituting a host number (shown in bold and italicized) for the host ID bits within that subnet. So, for example, host 2 in subnet 6 has 110 for the subnet ID and 00010 for the host ID, resulting in a final octet value of 11000010, or 194.

## **Class B Host Address Determination Example**

You can do the same thing for the Class B network, naturally. The address of that network is 166.113.0.0. Now say you want to define the hosts that go in subnet 13. You substitute 13 in binary (01101) for the subnet ID bits to get the following subnet address, which is shown with the subnet ID bits in bold and the host ID bits in italics:

10100110 01110001 **01101000** *00000000*

This is the subnet address 166.113.104.0. Now you have 11 bits of host ID, so you can have a maximum of 2,046 hosts. The first is found by substituting 000 00000001 for the host ID bits, which gives an address of 166.113.104.1. The second host is 166.113.104.2, and so on. The last is found by substituting 111 11111110, which gives an address of 166.113.111.254. Note that since the host ID bits extend over two octets, two octets change as you increment the host ID, unlike the Class C example. The broadcast address is 166.113.111.255.

**KEY CONCEPT** In a subnetted network, the address of Host H within subnet number S is found by plugging in the binary value of S for the network's subnet ID bits, and the binary value of H for the subnet's host ID bits.

## **Shortcuts for Computing Host Addresses**

As you can see, defining the host IDs is really quite straightforward. If you can substitute bits and convert to decimal, you have all you need to know. You can also see that, as was the case with defining the subnet addresses, there are patterns that you can use in defining host IDs and understanding how they work. These generally define ways for which you can more quickly determine certain host addresses by working directly in decimal instead of bothering with binary substitutions. This is a bit more complex conceptually, so proceed only if you are feeling a bit brave.

The following are some of the shortcuts you can use in determining host IP addresses in a subnet environment:

**First Host Address** *The first host address is always the subnet address with the last octet incremented by 1.* So in the Class C example, subnet 3's base address is 211.77.20.96. The first host address in subnet 3 is thus 211.77.20.97.

**Subsequent Host Addresses** After you find the first host address, to get the next one, you just add one to the last octet of the previous address. If this makes the last octet 256 (which can happen only if there are more than eight host ID bits), you “wrap around” this to zero and increment the third octet.

**Directly Calculating Host Addresses** If the number of host ID bits is eight or less, you can find host  $N$ 's address by adding  $N$  to the last octet's decimal value. For example, in the Class C example, subnet 3's base address is 211.77.20.96. Therefore, host 23 in this subnet has an address of 211.77.20.119. If there are more than eight bits in the host ID, this works for only the first 255 hosts, after which you need to wrap around and increase the value of the third octet. Consider again subnet 13 in the Class B example, which has a base address of 166.113.104.0. Host 214 on this subnet has address 166.113.104.0, but host 314 isn't 166.113.104.314. It is 166.113.105.58 (host 255 is 166.113.104.255, then host 256 is 166.113.105.0, and you count up 58 more (314–256) to get to 314, 166.113.105.58).

**Range of Host Addresses** For a range of hosts for any subnet, the first address is the base address of subnet with last octet incremented by one. The last address is the base address of *next subnet after this one*, less two in the last octet (which may require changing a 0 in the last octet to 254 and reducing the value of the third octet by 1). For example, consider subnet 17 in the Class B example. Its subnet address is 166.113.136.0. The address of subnet 18 is 166.113.144.0. So the range of hosts for subnet 17 is 166.113.136.1 to 166.113.143.254.

**Broadcast Address** *The broadcast address for a subnet is always one less than the base address of the subsequent subnet. Or alternatively, one more than the last real host address of the subnet.* So for subnet 17 in the Class B example, the broadcast address is 166.113.143.255.

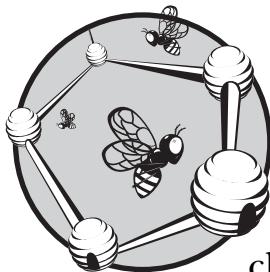
Did I just confuse you? Well, remember that these are shortcuts, and sometimes when you take a shortcut, you get lost. Just kidding; it's really not that hard once you play around with it a bit.

In closing, remember the following quick summary when working with IP addresses in a subnet environment:

- The network ID is the same for all hosts in all subnets and for all subnets in the network.
- The subnet ID is the same for all hosts in each subnet, but it's unique to each subnet in the network.
- The host ID is unique within each subnet. Each subnet has the same set of host IDs.
- Subnetting is fun! (Okay, okay, sorry. . . .)

# 20

## **IP CLASSLESS ADDRESSING— CLASSLESS INTER-DOMAIN ROUTING (CIDR)/SUPERNETTING**



As the Internet began to grow dramatically, three main problems arose with the original classful addressing scheme described in the previous chapters. These difficulties were addressed partially through subnet addressing, which provides more flexibility for the administrators of individual networks on an Internet. Subnetting, however, doesn't really tackle the problems in general terms. Some of these issues remain due to the use of classes even with subnets.

While development began on version 6 of the Internet Protocol (IPv6; see Part II-4) and its roomy 128-bit addressing system in the mid-1990s, developers recognized that it would take many years before widespread deployment of IPv6 would be possible. In order to extend the life of IPv4 until the newer version could be completed, it was necessary to take a new approach to addressing IPv4 devices. This new system calls for eliminating the notion of address classes entirely, creating a new *classless addressing* scheme sometimes called *Classless Inter-Domain Routing (CIDR)*.

In this chapter, I describe modern classless IP addressing. I begin with an overview of the concepts behind classless addressing and the idea behind *supernetting*, including why it was created and what its advantages and disadvantages are. I then define CIDR and describe how the system works in more detail, including the notation used for address blocks. I list each of the CIDR address block sizes and show how they relate to the older Class A, B, and C networks. I conclude with a CIDR addressing example that's similar to the examples in Chapter 19, but this one focuses on CIDR and is a bit more condensed.

## IP Classless Addressing and Supernetting Overview

Subnet addressing was an important development in the evolution of IP addressing, because it solved some important issues with the conventional, two-level class-based addressing scheme. Subnetting's contribution to flexibility in IP addressing was to allow each network to have its own two-level hierarchy, thereby giving the administrator of each network the equivalent of an Internet within the Internet.

When you looked at the advantages of subnetting in Chapter 18, you saw that subnetting was local within each organization and invisible to other organizations. This is an advantage in that it lets each organization tailor its network without other groups having to worry about the details of how this is done. Unfortunately, this invisibility also represents a key *disadvantage* of subnetted classful addressing: It cannot correct the fundamental inefficiencies associated with that type of addressing, because organizations are still assigned address blocks based on classes.

### The Main Problem with Classful Addressing

A key weakness of the subnetting system is its low granularity. A Class B address block contains a very large number of addresses (65,534), but a Class C block has only a relatively small number (254). There are many thousands of medium-sized organizations that need more than 254 IP addresses, but a small percentage of these needs 65,534 or anything even close to it. (The lack of a good match to a medium-sized organization with 5,000 hosts is illustrated in Figure 17-5 in Chapter 17.) When setting up their networks, these companies and groups would tend to request Class B address blocks and not Class C blocks, because they needed more than 254 hosts, without considering how many of the 65,000-odd addresses they really would use.

Due to how the classes of the older system were designed, there are over two million Class C address blocks, but only 16,384 Class B networks. While 16,384 seems like a lot at first glance, there are millions of organizations and corporations around the world. Class B allocations were being consumed at a rapid pace, while the smaller Class C networks were relatively unused.

The folks handing out Internet addresses needed a way to better utilize the address space so that it would not run out before the transition to IPv6. Subnetting didn't help a great deal with this problem. Why? Because it only works *within* the classful address blocks. If an organization needing 2,000 IP addresses requested a Class B block, they could use subnetting to more efficiently manage their block. However, subnetting could do nothing about the fact that this organization would never use over 62,000 of the addresses in its block—about 97 percent of their allocated address space.

The only solution to this would be to convince—or at worst case, force—companies to use many smaller Class C blocks instead of wasting the bulk of a Class B assignment. Many organizations resisted this due to the complexity involved, and this caused the other main problem that subnetting didn’t correct: the growth of Internet routing tables. Replacing one Class B network with 10 Class C networks means ten times as many entries for routers to maintain.

### ***The Solution: Eliminate Address Classes***

It was clear that as long as there were only three sizes of networks, the allocation efficiency problem could never be properly rectified. The solution was to get rid of the classes completely, in favor of a *classless* allocation scheme. This system would solve both of the main problems with classful addressing: inefficient address space use and the exponential growth of routing tables.

This system was developed in the early 1990s and formalized in 1993 in RFCs 1517, 1518, 1519, and 1520. The technology was called *Classless Inter-Domain Routing (CIDR)*. Despite this name, the scheme deals with both addressing and routing matters, since they are inextricably linked.

The idea behind CIDR is to adapt the concept of subnetting a single network to the entire Internet. In essence, classless addressing means that instead of breaking a particular network into subnets, you can aggregate networks into larger “supernets.” CIDR is sometimes called *supernetting* for this reason: It applies the principles of subnetting to larger networks. It is this aggregation of networks into supernets that allowed CIDR to resolve the problem of growing Internet routing tables.

Of course, if you are going to apply subnetting concepts to the entire Internet, you need to be able to have subnets of different sizes. After all, that’s one of the primary goals in eliminating the classes. So, more accurately, CIDR is an Internet-wide application of not just regular one-level subnetting, but of Variable Length Subnet Masking (VLSM), introduced in Chapter 18. Just as VLSM allows you split a network as many times as you want to create subnets, sub-subnets, and sub-sub-subnets, CIDR lets you do this with the entire Internet, as many times as needed.

**KEY CONCEPT** *Classless Inter-Domain Routing (CIDR)* is a system of IP addressing and routing that solves the many problems of classful addressing by eliminating fixed address classes in favor of a flexible, multiple-level, hierarchical structure of networks of varying sizes.

### ***The Many Benefits of Classless Addressing and Routing***

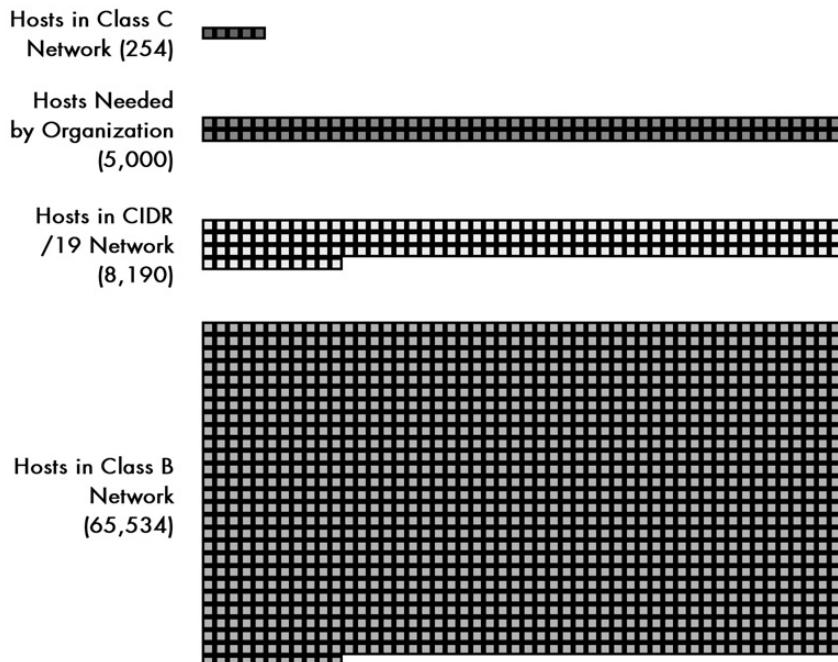
CIDR provides numerous advantages over the classful addressing scheme, whether or not subnetting is used:

**Efficient Address Space Allocation** Instead of allocating addresses in fixed-size blocks of low granularity, under CIDR, addresses are allocated in sizes of any binary multiple. So a company that needs 5,000 addresses can be assigned a block of 8,190 instead of 65,534, as shown in Figure 20-1. Or to think of it another way, the equivalent of a single Class B network can be shared among eight companies that each need 8,190 or fewer IP addresses.

**Elimination of Class Imbalances** There are no more Class A, B, and C networks, so there is no problem with some portions of the address space being widely used while others are neglected.

**Efficient Routing Entries** CIDR's multiple-level hierarchical structure allows a small number of routing entries to represent a large number of networks. Network descriptions can be aggregated and represented by a single entry. Since CIDR is hierarchical, the detail of lower-level, smaller networks can be hidden from routers that move traffic between large groups of networks. This is discussed more completely in Chapter 23, which covers IP routing issues.

**No Separate Subnetting Method** CIDR implements the concepts of subnetting within the Internet itself. An organization can use the same method used on the Internet to subdivide its internal network into subnets of arbitrary complexity, without needing a separate subnetting mechanism.



**Figure 20-1: Classless addressing (CIDR) solves the granularity problem** Figure 17-5 in Chapter 17 illustrates the primary problem with classful addressing: the great distance between the size of Class B and Class C networks. CIDR solves this issue by allowing any number of bits to be used for the network ID. In the case of an organization with 5,000 hosts, a /19 network with 8,190 hosts can be assigned. This reduces the address space waste for such an organization by about 95.

Since the main benefit of classful addressing was its simplicity, it's no surprise that the main drawback of CIDR is its greater complexity. One issue is that it is no longer possible to determine, by looking at the first octet, how many bits of an IP address represent the network ID and how many represent the host ID. A bit more care needs to be used in setting up routers as well, to make sure that routing is accomplished correctly.

## IP Supernetting: CIDR Hierarchical Addressing and Notation

When you first looked at IP addressing in Chapter 17, you saw that IP addresses were designed to be divided into a network identifier (network ID) and host identifier (host ID). Then, when subnets were introduced, you “stole” bits from the host ID to create a subnet ID, giving the IP address a total of three hierarchical levels. With VLSM, you further subnetted the subnets, taking more bits from the host ID to give you a multiple-level hierarchy with sub-subnets, sub-sub-subnets, and so forth.

In a classless environment, you completely change how you look at IP addresses by applying VLSM concepts not just to one network, but to the entire Internet. In essence, the Internet becomes just one giant network that is subnetted into a number of large blocks. Some of these large blocks are then broken down into smaller blocks, which can in turn be broken down further. This breaking down can occur multiple times, allowing you to split the “pie” of Internet addresses into slices of many different sizes to suit the needs of the organization.

As the name implies, classless addressing completely eliminates the prior notions of classes. There are no more Class A, B, and C blocks that are divided by the first few bits of the address. Instead, under CIDR, all Internet blocks can be of arbitrary size. Instead of having all networks use 8 (Class A), 16 (Class B), or 24 (Class C) bits for the network ID, you can have large networks with, say, 13 bits for the network ID (leaving 19 bits for the host ID), or very small ones that use 28 bits for the network ID (only 4 bits for the host ID). The size of the network is still based on the binary power of the number of host ID bits.

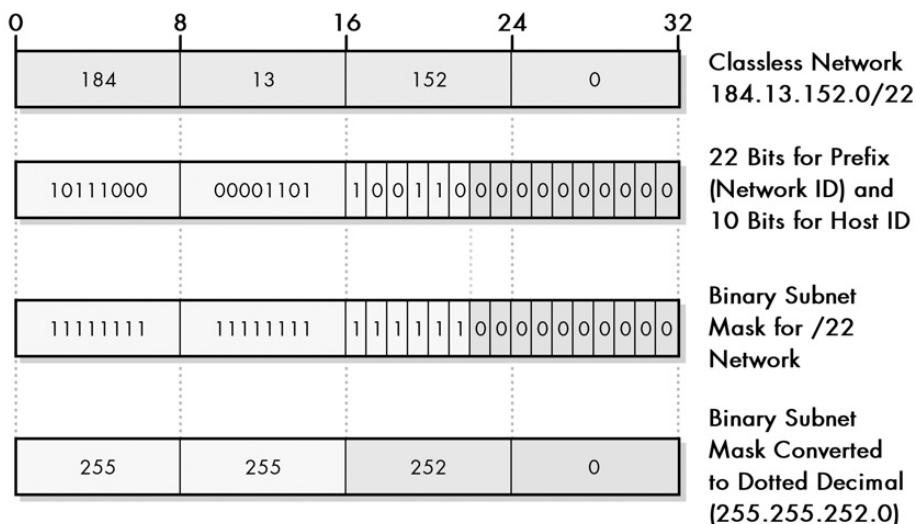
### CIDR (*Slash*) Notation

You’ll recall that when you used subnetting, you had a problem: Subnetting could be done by taking any number of available host ID bits, so how would devices know where the line was between the subnet ID and host ID? The same problem occurs under CIDR. There are no classes, so you can’t tell anything by looking at the first few bits of an IP address. Since addresses can have the dividing point between host ID and network ID occur anywhere, you need additional information in order to interpret IP addresses properly. Under CIDR, this impacts not only addresses within an organization, but also addresses in the entire Internet, since there are no classes and each network can be a different size.

For this reason, just as subnetting required the use of a subnet mask to show which bits belong to the network ID or subnet ID and which belong to the host ID, CIDR uses a subnet mask to show where the line is drawn between host ID and network ID. However, for simplicity, under CIDR you don’t usually work with 32-bit binary subnet masks. Instead, you use *slash notation*, more properly called *CIDR notation*. This notation shows the size of the network, sometimes called the *prefix length*, by following an IP address with an integer that tells you how many bits are used for the network ID (prefix).

**KEY CONCEPT** Since there are no address classes in CIDR, you cannot tell the size of the network ID of an address from the address alone. In CIDR, the length of the prefix (network ID) is indicated by placing it following a slash after the address. This is called *CIDR notation*, or *slash notation*.

For example, consider the network specification 184.13.152.0/22. The 22 means this network has 22 bits for the network ID and 10 bits for the host ID. This is equivalent to specifying a network with an address of 184.13.152.0 and a subnet mask of 255.255.252.0, as you can see in Figure 20-2. This sample network provides a total of 1,022 hosts ( $2^{10}-2$ ). The table in the following section shows all the different possible network sizes that can be configured under CIDR.



**Figure 20-2: CIDR (slash) notation and its subnet mask equivalent** A classless network is normally specified in CIDR, or slash notation, such as this example: 184.13.152.0/22. Here, the /22 means the first 22 bits of the address are the network ID. The equivalent subnet mask can be calculated by creating a 32-bit number with 22 ones followed by 10 zeros.

**NOTE** You may recall that under classful subnetting, the bits used for the subnet ID did not need to be contiguous. Even though this ability was almost never used to avoid confusion, noncontiguous subnet ID bits were possible. Under CIDR, the requirement for contiguous subnet ID bits has been made official—you could not use slash notation otherwise.

### Supernetting: Subnetting the Internet

In theory, then, what CIDR does is provide the central address-assignment authority with the flexibility to hand out address blocks of different sizes to organizations based on their need. However, when CIDR was developed, a shift was made in the method by which public IP addresses were assigned. Having everyone in the world attempt to get addresses from one organization wasn't the best method. It was necessary under the classful scheme because the hierarchy was only two levels deep. The Internet Assigned Numbers Authority (IANA) handed out network IDs to everyone, who then assigned host IDs (or subnetted).

Under CIDR, you have many hierarchical levels: You split big blocks into smaller blocks, and then still-smaller blocks, and so on. It makes sense to manage blocks in a similar hierarchical manner as well. So what happens is that IANA/ICANN divides addresses into large blocks, which it distributes to the four *regional Internet registries (RIRs)*: APNIC, ARIN, LACNIC, and RIPE NCC. These then further divide the address blocks and distribute them to lower-level national Internet registries (NIRs), local Internet registries (LIRs), and/or individual organizations such as Internet service providers (ISPs). This is all explained in the background discussion of Internet authorities and registries in Chapter 3.

ISPs can then divide these blocks into smaller ones and then allocate them to their customers. These customers are sometimes smaller ISPs themselves, which repeat the process. They split their blocks into pieces of different sizes and allocate them to their customers, some of whom are even smaller ISPs and some of whom are end users. The number of times this can occur is limited only by how many addresses are in the original block.

It's also worth noting that while CIDR is based on subnetting concepts, subnetting itself is not used in CIDR—or at least, not in the way it is used under classful addressing. There is no explicit subnetting using a subnet ID within CIDR. All IP addresses are interpreted only as having a network ID and a host ID. An organization does the equivalent of subnetting by dividing its own network into subnetworks using the same general method that ISPs do. This probably seems a bit confusing. Later in this chapter, I provide a detailed example of hierarchical address block assignments and how splitting works under CIDR.

### **Common Aspects of Classful and Classless Addressing**

There are a few aspects of addressing that were defined under the “classful” scheme that don’t change under CIDR:

**Private Address Blocks** Certain blocks of addresses are still reserved for private network addressing. These addresses are not directly routed on the Internet, but can be used in conjunction with Network Address Translation (NAT; see Chapter 28) to allow IP hosts without public addresses to access the Internet.

**Addresses with Special Meanings** The special meanings assigned to certain network ID and host ID patterns are the same as before. This is also why you still must subtract two from the number of hosts in each network. These represent the all-zeros case that refers to the network as a whole and the all-ones address used for broadcast.

**Loopback Addresses** The network 127.0.0.0 is still reserved for loopback functionality. (In CIDR it is given the notation 127.0.0.0/8.)

Finally, note that use of classless addressing requires hardware and software designed to handle it. If the hardware and software are still assuming that they are operating in a classful environment, they will not properly interpret addresses. Since CIDR has now been around for more than a decade, this is usually not a problem with modern systems.

## IP Classless Addressing Block Sizes and Classful Network Equivalents

Because CIDR allows you to divide IP addresses into network IDs and host IDs along any bit boundary, it allows for the creation of dozens of different sizes of networks. As with subnetting, the size of network is a trade-off between the number of bits used for the network ID and the number used for the host ID. Unlike conventional subnetting, where a single choice is made for all subnets, CIDR allows many levels of hierarchical division of the Internet, so many sizes of networks exist simultaneously. Larger networks are created and subdivided into smaller ones.

Since many people are used to looking at IP address blocks in terms of their classful sizes, it is common to express CIDR address blocks in terms of their classful equivalents. First, at this point it should be simple to see that a CIDR /8 network is equal in size to a Class A network, a /16 is equivalent to a Class B network, and a /24 is equivalent to a Class C network. This is because Class A networks use 8 bits for the network ID, Class B networks use 16, and Class C networks use 24. However, remember that these CIDR equivalents do not need to have any particular ranges for their first octets as in the classful scheme.

Each time you reduce the prefix length, you are defining a network about double the size of the one with the higher number, since you have increased the number of bits in the host ID by one. So, a /15 network is equal in size to two /16 networks.

Table 20-1 shows each of the possible theoretical ways to divide the 32 bits of an IP address into network ID and host ID bits under CIDR. For each, I have shown the number of hosts in each network, and the way a network of each size is represented in both slash notation and as a conventional subnet mask. I have also shown the equivalent number of Class A, Class B, and Class C networks for each.

Keep the following things in mind while looking at this table:

- Some of the entries shown are more theoretical than practical and are included merely for completeness. This is particularly the case with the larger networks. For example, I doubt anyone ever actually works with a /1 or /2 CIDR network; there would be only two of the former and four of the latter encompassing the entire IP address space! Most of the time, you will be working with smaller networks, /16 and below.
- Under normal circumstances, you cannot have a /31 or /32 CIDR network, because it would have zero valid host IDs. (There is a special case: /31 networks can be used for point-to-point links, where it is obvious who the intended recipient is of each transmission, and where broadcasts are not necessary. This is described in RFC 3021.)
- In the columns showing the number of equivalent Class A, B, and C networks I have only shown numbers in the range of 1/256 to 256 for simplicity. Obviously, a /6 network, in addition to being equal in size to four Class A networks, also equals 1,024 Class B networks, and 262,144 Class C networks, but few people would bother referring to a /6 as being 262,144 Class C networks.

**Table 20-1:** CIDR Address Blocks and Classful Address Equivalents

# of Bits for Network ID	# of Bits for Host ID	# of Hosts per Network	Prefix Length in Slash Notation	Equivalent Subnet Mask	# of Equivalent Classful Addressing Networks		
					Class A	Class B	Class C
1	31	2,147,483,646	/1	128.0.0.0	128	—	—
2	30	1,073,741,822	/2	192.0.0.0	64	—	—
3	29	536,870,910	/3	224.0.0.0	32	—	—
4	28	268,435,454	/4	240.0.0.0	16	—	—
5	27	134,217,726	/5	248.0.0.0	8	—	—
6	26	67,108,862	/6	252.0.0.0	4	—	—
7	25	33,554,430	/7	254.0.0.0	2	—	—
8	24	16,777,214	/8	255.0.0.0	1	256	—
9	23	8,388,606	/9	255.128.0.0	1/2	128	—
10	22	4,194,302	/10	255.192.0.0	1/4	64	—
11	21	2,097,150	/11	255.224.0.0	1/8	32	—
12	20	1,048,574	/12	255.240.0.0	1/16	16	—
13	19	524,286	/13	255.248.0.0	1/32	8	—
14	18	262,142	/14	255.252.0.0	1/64	4	—
15	17	131,070	/15	255.254.0.0	1/128	2	—
16	16	65,534	/16	255.255.0.0	1/256	1	256
17	15	32,766	/17	255.255.128.0	—	1/2	128
18	14	16,382	/18	255.255.192.0	—	1/4	64
19	13	8,190	/19	255.255.224.0	—	1/8	32
20	12	4,094	/20	255.255.240.0	—	1/16	16
21	11	2,046	/21	255.255.248.0	—	1/32	8
22	10	1,022	/22	255.255.252.0	—	1/64	4
23	9	510	/23	255.255.254.0	—	1/128	2
24	8	254	/24	255.255.255.0	—	1/256	1
25	7	126	/25	255.255.255.128	—	—	1/2
26	6	62	/26	255.255.255.192	—	—	1/4
27	5	30	/27	255.255.255.224	—	—	1/8
28	4	14	/28	255.255.255.240	—	—	1/16
29	3	6	/29	255.255.255.248	—	—	1/32
30	2	2	/30	255.255.255.252	—	—	1/64

## IP CIDR Addressing Example

The multiple hierarchical levels of CIDR make the technology seem rather complicated. However, understanding how CIDR works really is not that difficult, assuming you already know how subnetting is done. In particular, if you know how VLSM functions, you basically already know how CIDR works, since they are pretty much the same thing. They differ only in the way that the hierarchical division of networks is accomplished, and in the terminology.

To show how CIDR works better, let's take an example that will illustrate the power of classless addressing: its ability to selectively subdivide a large block of addresses into smaller ones that suit the needs of various organizations. Since address allocation in CIDR typically starts with larger blocks owned by larger ISPs, let's start there as well.

Suppose you have an ISP that is just starting up. It's not a major ISP, but a moderate-sized one with only a few customers, so it needs only a relatively small allocation. It begins with the block 71.94.0.0/15. The /15 on the end of the block address tells you that this is a block of addresses where the first 15 bits are the network ID and the last 17 are the host ID. This block was obtained from a larger ISP, carved from a larger block of addresses by that ISP. For example, 71.94.0.0/15 would be equal to half of the address block 71.92.0.0/14, a quarter of the block 71.88.0.0/13, and so on.

The ISP's block is equal in size to two Class B networks and has a total of 131,070 possible host addresses. This ISP can choose to divide this block in a variety of ways, depending on the needs of its clients and its own internal use. However, this ISP is just starting up, so it is not even sure of what its ultimate needs will be. Let's say it expects to resell about half of its address space to other ISPs, but isn't sure what sizes they will need yet. Of the other half, it plans to split it into four different sizes of blocks to match the needs of different-sized organizations.

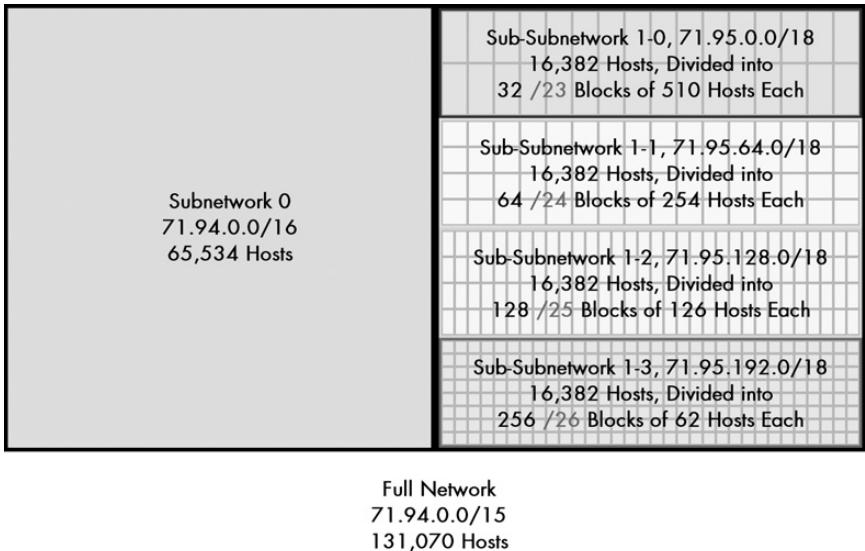
To imagine how the ISP divides its address space, you can consider the analogy of cutting up a pie. The ISP will first cut the pie in half and reserve one-half for its future ISP customers. It will then cut the other half into some large pieces and some small pieces. This is illustrated in Figure 20-3. (Okay, I know it's a square pie. I wanted to show the individual small blocks to scale.)

The actual process of division might follow the progression described in the following section and illustrated in Figure 20-4.

### ***First Level of Division***

The “pie” is initially cut down the middle by using the single leftmost host ID bit as an extra network bit. Here's the network address block, 71.94.0.0/15 in binary, with the leftmost host ID bit shown in bold:

01000111 0101111**0** 00000000 00000000



**Figure 20-3: Example of a hierarchical division of a /15 CIDR address block** This diagram shows one method by which an ISP with a relatively large /15 address block (131,070 hosts) might choose to hierarchically divide it. In this case it is first divided in half into two /16 blocks. One is reserved, while the other is divided into four /18 blocks. Each of those is divided into blocks of a different size to allow allocation to organizations requiring up to 62, 126, 254, or 510 hosts, respectively.

To make the split, you make one network equal to this binary network address with the highlighted bit remaining zero, and the other one with it changed to a one. This creates two subnetworks—not subnets as in the classful sense of the word, but portions of the original network—that I have numbered based on the numeric value of what is substituted into the new network ID bits, as follows:

Subnetwork 0: 01000111 0101111**0** 00000000 00000000

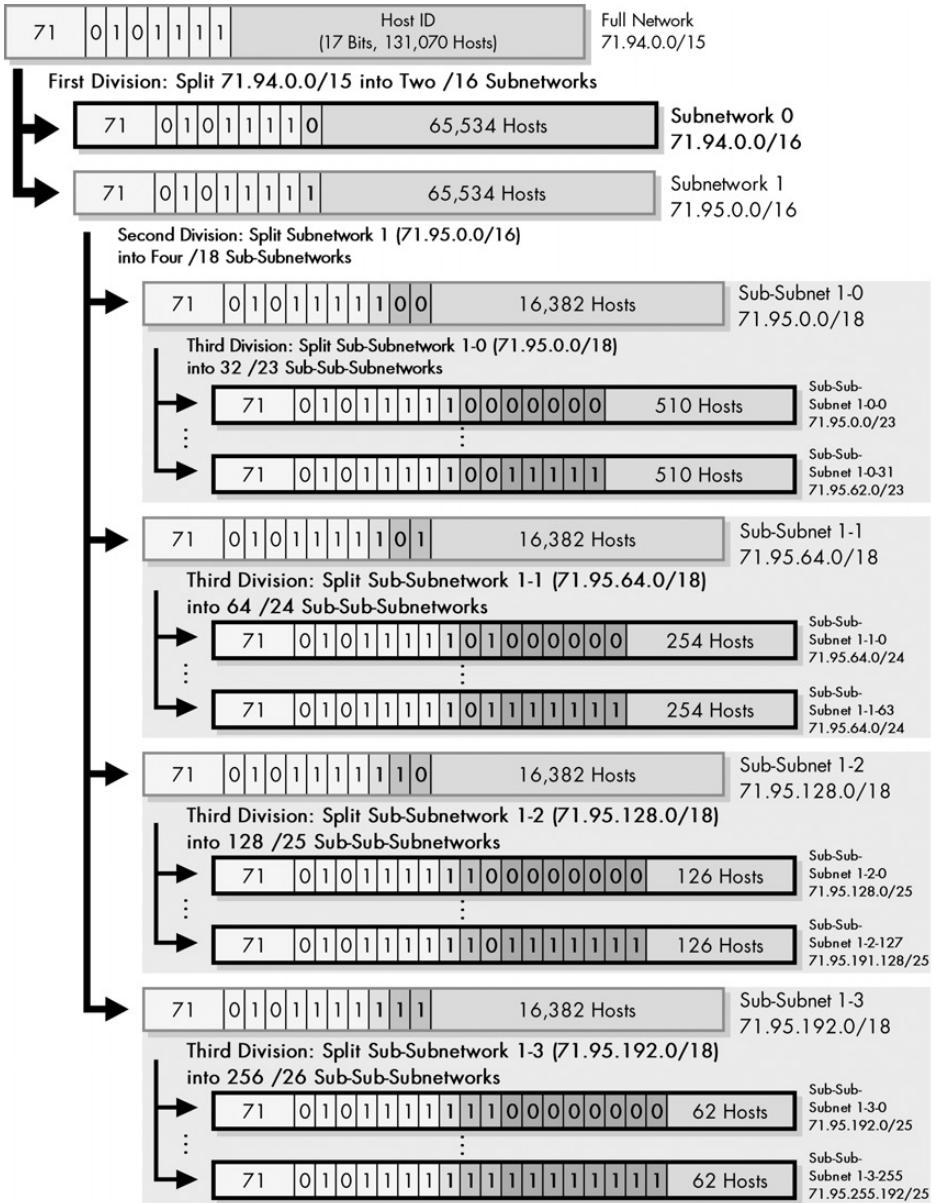
Subnetwork 1: 01000111 0101111**1** 00000000 00000000

Because bit 16 is now also part of the network address, these are /16 networks, the size of a classful Class B network. So the subnetworks are as follows:

Subnetwork 0: 71.94.0.0/16

Subnetwork 1: 71.95.0.0/16

You'll notice subnetwork 0 has the same IP address as the larger network it came from; this is always true of the subnetwork 0 in a network.



**Figure 20-4:** Hierarchical address division using CIDR

## ***Second Level of Division***

Let's say you set aside subnetwork 0 earlier for future ISP allocations. You then choose to divide the second subnetwork into four. These you will then further subdivide into different sizes to meet the customer's needs. To divide into four groups, you need two more bits from the host ID of subnetwork 1, as shown here in bold and underlined next to the original subnet bit:

01000111 01011111 **00**000000 00000000

These two bits are replaced by the patterns 00, 01, 10, and 11 to get four sub-subnetworks. They will be /18 networks, since you took two extra bits from the host ID of a /16 as shown here:

Sub-subnetwork 1-0: 01000111 01011111 **00**000000 00000000 (71.95.0.0/18)

Sub-subnetwork 1-1: 01000111 01011111 **01**000000 00000000 (71.95.64.0/18)

Sub-subnetwork 1-2: 01000111 01011111 **10**000000 00000000 (71.95.128.0/18)

Sub-subnetwork 1-3: 01000111 01011111 **11**000000 00000000 (71.95.192.0/18)

Each of these has 16,382 addresses.

## ***Third Level of Division***

You now take each of the four /18 networks and further subdivide it. You want to make each of these contain a number of blocks of different sizes corresponding to the potential customers. One way to do this would be as follows:

**Larger Organizations** Customers needing up to 510 addresses require a /23 network. You divide sub-subnetwork 1-0, 71.95.0.0/18 by taking five bits from the host ID field:

01000111 01011111 **00000**00 00000000

You substitute into these five bits 00000, 00001, 00010 and so on, giving you 32 different /23 networks in this block, each containing nine bits for the host ID, for 510 hosts. The first will be sub-sub-subnetwork 1-0-0, 71.95.0.0/23; the second sub-sub-subnetwork 1-0-1, 71.95.2.0/23; the last will be sub-sub-subnetwork 1-0-31: 71.95.62.0/23.

**Medium-Sized Organizations** For customers needing up to 254 addresses, you divide sub-subnetwork 1-1, 71.95.64.0/18, by taking six bits from the host ID field:

01000111 01011111 **010000**00 00000000

This gives you 64 different /24 networks. The first will be sub-sub-subnetwork 1-1-0, 71.95.64.0/24, the second sub-sub-subnetwork 1-1-1, 71.95.65.0/24, and so on.

**Smaller Organizations** For customers with up to 126 hosts, you divide sub-subnetwork 1-2, 71.95.128.0/18, by taking seven bits from the host ID field, as follows:

01000111 01011111 **10000000 00000000**

Seven bits allow 128 of these /25 networks within the /18 block. The first will be 71.95.128.0/25, the second 71.95.128.128/25, the third 71.95.129.0/25, and so on.

**Very Small Organizations** For customers with up to 60 hosts, you divide sub-subnetwork 1-3, 71.95.192.0/18, by taking eight bits from the host ID field:

01000111 01011111 **11000000 00000000**

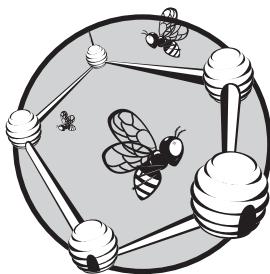
This gives you 256 different /26 networks within the /18 block. The first will be 71.95.192.0/26, the second 71.95.192.64/26, and so on.

This example shows only one of many different ways to slice up this pie. The ISP might decide that creating four different sizes of customer networks in advance was not the right way to go. It might instead just take the tack of dividing the pie in half, dividing it in half again, and so on, as many times as needed to create slices of the right size. Alternatively, if most of their customers needed around 50, 100, 200, or 500 hosts, the previous example might be the easiest to administer.

It would still be possible for the ISP to divide any of the smaller blocks further if they needed to do so. They could split a /26 sub-sub-subnetwork into four /28 sub-sub-sub-subnetworks for very small customers, for example. Also, an individual customer of this ISP could do the same thing, dividing its own block to suit the internal structure of its network.

# 21

## INTERNET PROTOCOL DATAGRAM ENCAPSULATION AND FORMATTING



The primary job of the Internet Protocol (IP) is to deliver data between devices over an internetwork. On its journey between two hosts in an internetwork, this data may travel across many physical networks. To help ensure that the data is sent and received properly, it is *encapsulated* within a message called an *IP datagram*. This datagram includes several fields that help manage the operation of IP and ensure that data gets where it needs to go.

In this chapter, I take a look at how IP takes data passed to it from higher layers and packages it for transmission. I begin with a general discussion of IP datagrams and encapsulation. I then describe the general format of IP datagrams, including the fields used in the IP header and how they are interpreted. I also include a brief discussion of IP datagram options and their use.

**BACKGROUND INFORMATION** *This chapter assumes at least passing familiarity with IP addressing concepts, as outlined in Chapters 16–20. It also makes reference to the chapter on datagram fragmentation and reassembly (Chapter 22).*

**NOTE** IP datagrams are sometimes called IP packets. Whether datagram or packet is the preferred term seems to depend on whom you ask; even the standards don't use one term exclusively. On the other hand, I have seen IP datagrams called IP frames, and that's definitely not correct! Chapter 1 describes these terms more completely.

## IP Datagram Encapsulation

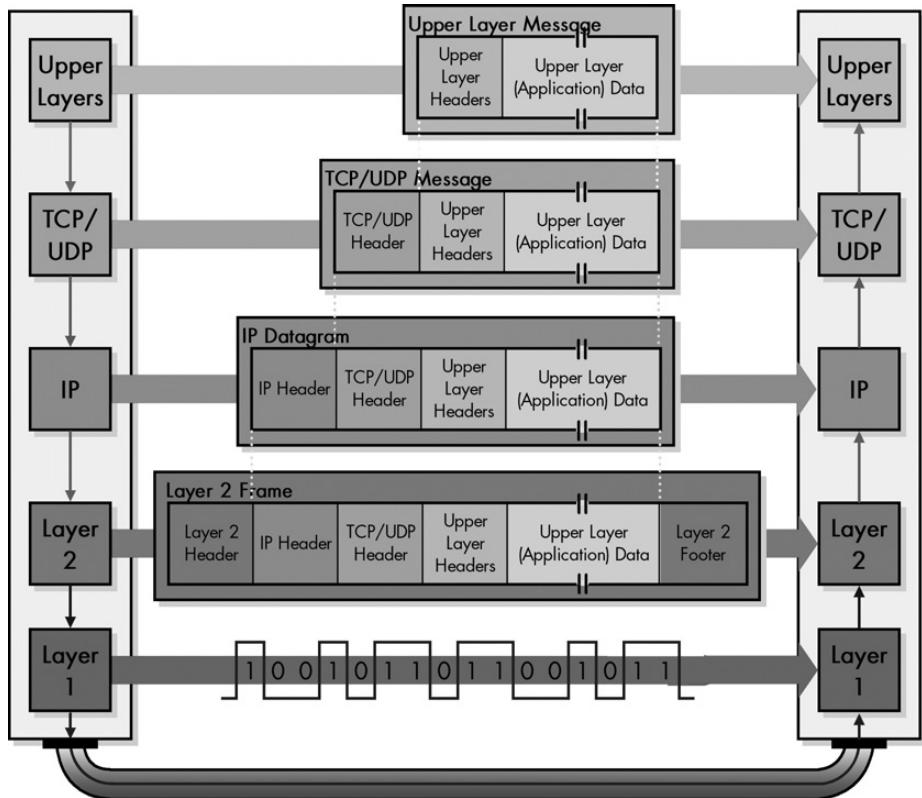
In Chapter 5, which described OSI Reference Model concepts, I looked at several ways that protocols at various layers in a networking protocol stack interact with each other. One of the most important concepts in interprotocol operation is that of *encapsulation*. Most data originates within the higher layers of the OSI model. The protocols at these layers pass the data down to lower layers for transmission, usually in the form of discrete messages. Upon receipt, each lower-level protocol takes the entire contents of the message received and encapsulates it into its own message format, adding a header and possibly a footer that contain important control information.

You might think of encapsulation as similar to sending a letter enclosed in an envelope. You write a letter and put it in an envelope with a name and address, but if you give it to a courier for overnight delivery; the courier takes that envelope and puts it in a larger delivery envelope. In a similar way, messages at higher networking layers are encapsulated in lower-layer messages, which can then in turn be further encapsulated.

Due to the prominence of TCP/IP, IP is one of the most important places where data encapsulation occurs on a modern network. Data is passed to IP typically from one of the two main transport layer protocols: the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). This data is already in the form of a TCP or UDP message with TCP or UDP headers. This is then encapsulated into the body of an IP message, usually called an *IP datagram* or *IP packet*. Encapsulation and formatting of an IP datagram is also sometimes called *packaging*—again, the envelope is an obvious comparison.

Figure 21-1 displays this entire process, which looks very similar to the drawing of the OSI Reference Model as a whole, as shown in Figure 5-5 in Chapter 5. As you can see, an upper-layer message is packaged into a TCP or UDP message. This then becomes the payload of an IP datagram, shown here with only one header (things can get a bit more complex than this). The IP datagram is then passed down to layer 2, where it is encapsulated into some sort of local area network (LAN), wide area network (WAN), or wireless LAN (WLAN) frame, and then converted to bits and transmitted at the physical layer.

If the message to be transmitted is too large to pass through the underlying network, it may first be fragmented. This is analogous to splitting up a large delivery into multiple smaller envelopes or boxes. In this case, each IP datagram carries only part of the higher-layer message. The receiving device must reassemble the message from the IP datagrams.



**Figure 21-1: IP datagram encapsulation** The upper-layer message is packaged into a TCP or UDP message, which becomes the payload of an IP datagram. The IP datagram is then passed down to layer 2, where it is encapsulated in a LAN, WAN, or WLAN frame. It is then converted to bits and transmitted at the physical layer.

The IP datagram is somewhat similar in concept to a frame used in Ethernet or another data link layer, except that IP datagrams are designed to facilitate transmission across an internetwork, while data link layer frames are used only for direct delivery within a physical network. The fields included in the IP header are used to manage internetwork datagram delivery. This includes key information for delivery, such as the address of the destination device, identification of the type of frame, and control bits. The header follows a format that you will examine shortly.

Once data is encapsulated into an IP datagram, it is passed down to the data link layer for transmission across the current “hop” of the internetwork. There it is further encapsulated, IP header and all, into a data link layer frame such as an Ethernet frame. An IP datagram may be encapsulated into many such data link layer frames as it is routed across the internetwork; on each hop, the IP datagram is removed from the data link layer frame and then repackaged into a new one for the next hop. The IP datagram, however, is not changed (except for some control fields) until it reaches its final destination.

## IP Datagram General Format

Data transmitted over an internetwork using IP is carried in messages called *IP datagrams*. As is the case with all network protocol messages, IP uses a specific format for its datagrams. Here, I will discuss the IP version 4 (IPv4) datagram format, which was defined in RFC 791 along with the rest of IPv4.

The IPv4 datagram is conceptually divided into two pieces: the *header* and the *payload*. The header contains addressing and control fields, while the payload carries the actual data to be sent over the internetwork. Unlike some message formats, IP datagrams do not have a footer following the payload.

Even though IP is a relatively simple, connectionless, and unreliable protocol, the IPv4 header carries a fair bit of information, which makes it rather large. It is at least 20 bytes long, and with options it can be significantly longer. The IP datagram format is described in Tables 21-1, 21-2, and 21-3, and illustrated in Figure 21-2.

**Table 21-1:** Internet Protocol Version 4 (IPv4) Datagram Format

Field Name	Size (Bytes)	Description
Version	1/2 (4 bits)	Identifies the version of IP used to generate the datagram. For IPv4, this is the number 4. This field ensures compatibility between devices that may be running different versions of IP. In general, a device running an older version of IP will reject datagrams created by newer implementations, under the assumption that the older version may not be able to interpret the newer datagram correctly.
IHL	1/2 (4 bits)	Specifies the length of the IP header, in 32-bit words. This includes the length of any options fields and padding. The normal value of this field when no options are used is 5 (5 32-bit words = $5 \times 4 = 20$ bytes). Contrast this with the longer Total Length field in this table.
TOS	1	A field designed to carry information to provide quality-of-service features, such as prioritized delivery for IP datagrams. This has not been as widely used as originally defined, and its meaning has been redefined for use by a technique called <i>Differentiated Services (DS)</i> , as discussed in the “IP Datagram Type of Service (TOS) Field” section of this chapter.
TL	2	Specifies the total length of the IP datagram, in bytes. Since this field is 16 bits wide, the maximum length of an IP datagram is 65,535 bytes, though most are much smaller.
Identification	2	This field contains a 16-bit value that is common to each of the fragments belonging to a particular message; for datagrams originally sent unfragmented, it is still filled in so it can be used if the datagram must be fragmented by a router during delivery. The recipient uses this field to reassemble messages without accidentally mixing fragments from different messages. This is needed because fragments may arrive from multiple messages mixed together, since IP datagrams can be received out of order from any device. (See the discussion of IP message fragmentation in Chapter 22.)
Flags	3/8 (3 bits)	Three control flags, two of which are used to manage fragmentation (as described in the topic on fragmentation), and one that is reserved. See Table 21-2.
Fragment Offset	1 5/8 (13 bits)	When fragmentation of a message occurs, this field specifies the offset, or position, in the message where the data in this fragment goes in units of eight bytes (64 bits). The first fragment has an offset of 0. (See the discussion of fragmentation in Chapter 27 for a description of how the field is used.)
TTL	1	This specifies how long the datagram is allowed to live on the network, in router hops. Each router decrements the value of the TTL field (reduces it by one) prior to transmitting it. If the TTL field drops to zero, the datagram is assumed to have taken too long a route and is discarded. (See the “IP Datagram Time to Live (TTL) Field” section later in this chapter for more information.)

(continued)

**Table 21-1:** Internet Protocol Version 4 (IPv4) Datagram Format (continued)

Field Name	Size (Bytes)	Description
Protocol	1	Identifies the higher-layer protocol (generally either a transport layer protocol or encapsulated network layer protocol) carried in the datagram. Table 21-3 shows the protocol values of this field, which were originally defined by the IETF “Assigned Numbers” standard, RFC 1700, and are now maintained by the Internet Assigned Numbers Authority (IANA).
Header Checksum	2	A checksum is computed over the header to provide basic protection against corruption in transmission. This is not the more complex cyclic redundancy check (CRC) code that’s typically used by data link layer technologies such as Ethernet; it’s just a 16-bit checksum. It is calculated by dividing the header bytes into words (a word is two bytes) and then adding them together. Only the header is checksummed; not the data. At each hop, the device receiving the datagram does the same checksum calculation, and if there is a mismatch, it discards the datagram as damaged.
Source Address	4	This is the 32-bit IP address of the originator of the datagram. Note that even though intermediate devices such as routers may handle the datagram, they do not normally put their address into this field—the address is always that of the device that originally sent the datagram.
Destination Address	4	This is the 32-bit IP address of the intended recipient of the datagram. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always used to specify the ultimate destination.
Options	Variable	One or more of several types of options may be included after the standard headers in certain IP datagrams, as discussed later in this chapter, in the “IP Datagram Options and Option Format” section.
Padding	Variable	If one or more options are included, and the number of bits used for them is not a multiple of 32, enough 0 bits are added to pad out the header to a multiple of 32 bits (four bytes).
Data	Variable	This is the data that will be transmitted in the datagram. It is either an entire higher-layer message or a fragment of one.

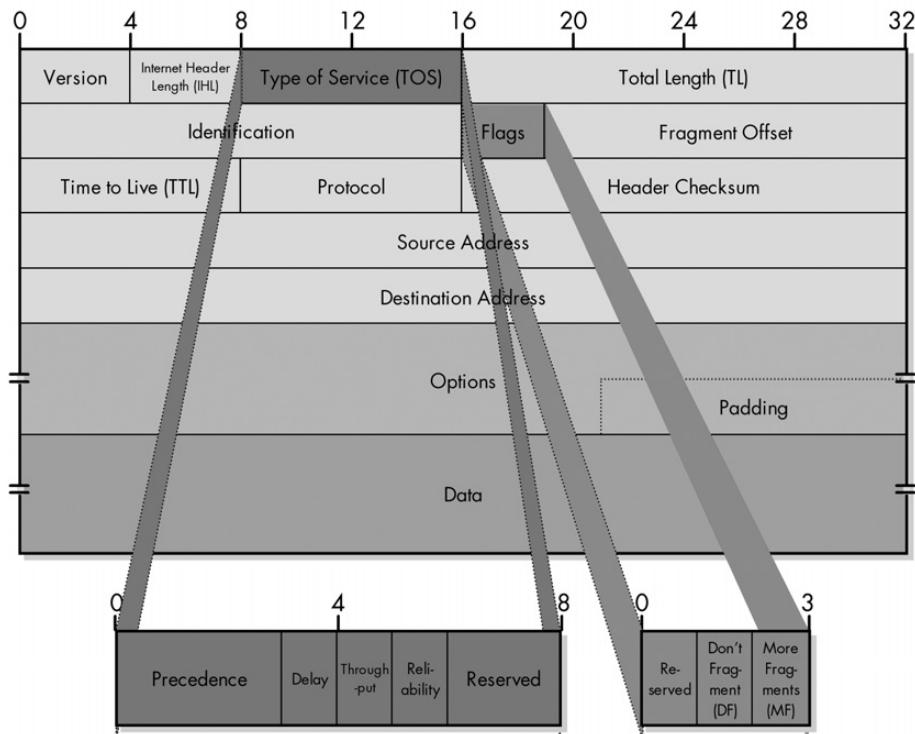
**Table 21-2:** IPv4 Flags Subfields

Subfield Name	Size (Bytes)	Description
Reserved	1/8 (1 bit)	Not used.
DF	1/8 (1 bit)	When set to 1, this says that the datagram should not be fragmented. Since the fragmentation process is generally invisible to higher layers, most protocols don’t care about this and don’t set this flag. It is, however, used for testing the maximum transmission unit (MTU) of a link.
MF	1/8 (1 bit)	When set to 0, this indicates the last fragment in a message; when set to 1, it indicates that more fragments are yet to come in the fragmented message. If no fragmentation is used for a message, there is only one fragment (the whole message), and this flag is 0. If fragmentation is used, all fragments but the last set this flag to 1 so that the recipient knows when all fragments have been sent.

**Table 21-3: IPv4 Protocol Subfields**

Value (Hexadecimal)	Value (Decimal)	Protocol
00	0	Reserved
01	1	ICMP
02	2	IGMP
03	3	GGP
04	4	IP-in-IP Encapsulation
06	6	TCP
08	8	EGP
11	17	UDP
32	50	Encapsulating Security Payload (ESP) Extension Header
33	51	Authentication Header (AH) Extension Header

**NOTE** The last two entries in Table 21-3 are used when IPSec inserts additional headers into the datagram: the AH or ESP headers. See Chapter 29 for more information.



**Figure 21-2: IPv4 datagram format** This diagram shows the all-important IPv4 datagram format. The first 20 bytes are the fixed IP header, followed by an optional Options section, and a variable-length Data area. Note that the Type of Service field is shown as originally defined in the IPv4 standard.

## **IP Datagram Time to Live (TTL) Field**

Let's look at the Time to Live (TTL) field. Since IP datagrams are sent from router to router as they travel across an internetwork, a datagram could be passed from Router A to Router B to Router C, and then back to Router A. This is called a *router loop*, and it's something that we don't want to happen.

To ensure that datagrams don't circle around endlessly, the TTL field was designed to contain a time value (in seconds), which would be filled in when the datagram was originally sent. Routers would decrease the time value periodically, and if it ever hit zero, destroy the datagram. The TTL field was also designed to ensure that time-critical datagrams wouldn't become stale or pass their expiration date.

In practice, this field is not used in exactly this manner. Routers today are fast and usually take far less than a second to forward a datagram, which makes it impractical to measure the time that a datagram lives. Instead, this field is used as a maximum hop count for the datagram. Each time a router processes a datagram, it reduces the value of the TTL field by one. If doing this results in the field being zero, the datagram is said to have expired, at which point it is dropped, and usually an Internet Control Message Protocol (ICMP) Time Exceeded message is sent to inform the originator of the message that it has expired. The TTL field is one of the primary mechanisms by which networks are protected from router loops. (See the description of ICMP Time Exceeded messages in Chapter 32 for more on how TTL helps IP handle router loops.)

## **IP Datagram Type of Service (TOS) Field**

The Type of Service (TOS) field is a one-byte field that was originally intended to provide certain quality-of-service (QoS) features for IP datagram delivery. It allowed IP datagrams to be tagged with information indicating not only their precedence, but also the preferred manner in which they should be delivered. It was divided into a number of subfields, as shown in Table 21-4 and Figure 21-2.

The lack of QoS features has been considered a weakness of IP for a long time. But as you can see in Table 21-4, these features were built into IP from the start. The fact is that even though this field was defined in the standard in the early 1980s, it was not widely used by hardware and software. For years, it was just passed around with all zeros in the bits and mostly ignored.

The Internet Engineering Task Force (IETF), seeing the field unused, attempted to revive its use. In 1998, RFC 2474 redefined the first six bits of the TOS field to support a technique called *Differentiated Services (DS)*. Under DS, the values in the TOS field are called *codepoints* and are associated with different service levels. (See RFC 2474 for all the details.)

**RELATED INFORMATION** Be sure to read the remainder of this chapter for more information on how IP options are used in datagrams and Chapter 22 for some more context on the use of fragmentation-related fields such as *Identification*, *Fragment Offset*, and *More Fragments*.

**Table 21-4:** Original Definition of IPv4 Type of Service (TOS) Field

Subfield Name	Size (Bytes)	Description
Precedence	3/8 (3 bits)	A field indicating the priority of the datagram. There were eight defined values, from lowest to highest priority: 000: Routine 001: Priority 010: Immediate 011: Flash 100: Flash Override 101: CRITIC/ECP 110: Internetwork Control 111: Network Control
D	1/8 (1 bit)	Set to 0 to request normal delay in delivery; set to 1 if a low delay delivery is requested.
T	1/8 (1 bit)	Set to 0 to request normal delivery throughput; set to 1 if higher throughput delivery is requested.
R	1/8 (1 bit)	Set to 0 to request normal reliability in delivery; set to 1 if higher reliability delivery is requested.
Reserved	2/8 (2 bits)	Not used.

## IP Datagram Options and Option Format

All IP datagrams must include the standard 20-byte header that contains key information such as the source and destination address of the datagram, fragmentation control parameters, length information, and more. In addition to these invariable fields, the creators of IPv4 included the ability to add *options* that provide additional flexibility in how IP handles datagrams. Use of these options is, of course, optional. However, all devices that handle IP datagrams must be capable of properly reading and handling them.

The IP datagram may contain zero, one, or more options, so the total length of the Options field in the IP header is variable. Each of the options can be a single byte or multiple bytes in length, depending on how much information the option needs to convey. When more than one option is included, they are concatenated and put into the Options field as a whole. Since the IP header must be a multiple of 32 bits, a Padding field is included if the number of bits in all options together is not a multiple of 32 bits.

Each IP option has its own subfield format, generally structured as shown in Tables 21-5 and 21-6, and illustrated in Figure 21-3. For most options, all three subfields are used: Option Type, Option Length, and Option Data. For a few simple options, however, this complex substructure is not needed. In those cases, the option type itself communicates all the information required, so the Option Type field appears, and the Option Length and Option Data subfields are omitted.

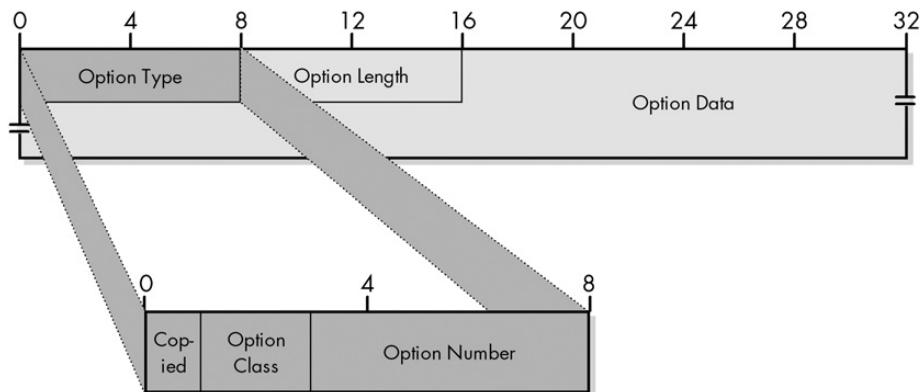
**Table 21-5:** IPv4 Option Format

Subfield Name	Size (Bytes)	Description
Option Type	1	The Option Type subfield is divided into three subsubfields, as shown in Table 21-6.
Option Length	0 or 1	For variable-length options, indicates the size of the entire option, including all three subfields shown here, in bytes.
Option Data	0 or variable	For variable-length options, contains data to be sent as part of the option.

**Table 21-6:** IPv4 Options: Option Type Subfields

Sub-Subfield Name	Size (Bytes)	Description
Copied Flag	1/8 (1 bit)	This bit is set to 1 if the option is intended to be copied into all fragments when a datagram is fragmented; it is cleared to 0 if the option should not be copied into fragments.
Option Class	2/8 (2 bits)	Specifies one of four potential values that indicate the general category into which the option belongs. In fact, only two of the values are used: 0 is for Control options, and 2 for Debugging and Measurement.
Option Number	5/8 (5 bits)	Specifies the kind of option. 32 different values can be specified for each of the two option classes. Of these, a few are more commonly employed. See Table 21-7 for more information on the specific options.

Table 21-7 lists the most common IPv4 options, showing the option class, option number, and length for each (a length of 1 indicates that an option consists of only an Option Type field). The table also provides a brief description of how each is used.



**Figure 21-3: IPv4 Options field format** This diagram shows the full field format for an IPv4 option. Note that a few simple options may consist of only the Option Type subfield, with the Option Length and Option Data subfields omitted.

**Table 21-7:** Common IPv4 Options

Option Class	Option Number	Length (Bytes)	Option Name	Description
0	0	1	End of Options List	An option containing just a single zero byte, used to mark the end of a list of options.
0	1	1	No Operation	A “dummy option” used as internal padding to align certain options on a 32-bit boundary when required.
0	2	11	Security	An option provided for the military to indicate the security classification of IP datagrams.
0	3	Variable	Loose Source Route	One of two options for source routing of IP datagrams.
0	7	Variable	Record Route	Allows the route used by a datagram to be recorded within the header for the datagram itself. If a source device sends a datagram with this option in it, each router that handles the datagram adds its IP address to this option. The recipient can then extract the list of IP addresses to see the route taken by the datagram. Note that the length of this option is set by the originating device. It cannot be enlarged as the datagram is routed, and if it fills up before it arrives at its destination, only a partial route will be recorded.
0	9	Variable	Strict Source Route	One of two options for source routing of IP datagrams.
2	4	Variable	Timestamp	Works similar to the Record Route option, but each device puts in a timestamp, so the recipient can see how long it took for the datagram to travel between routers. As with the Record Route option, the length of this option is set by the originating device and cannot be enlarged by intermediate devices.
2	18	12	Traceroute	Used in the enhanced implementation of the traceroute utility, as described in RFC 1393. Also see Chapter 33, which discusses ICMP traceroute messages.

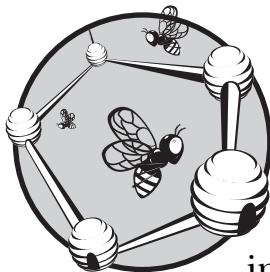
**KEY CONCEPT** Each IPv4 datagram has a 20-byte mandatory header and may also include one or more *options*. Each option has its own field format, and most are variable in size.

Normally, IP datagrams are routed without any specific instructions from devices about the path a datagram should take from the source to the destination. It’s the job of routers to use routing protocols and to figure out those details. In some cases, however, it may be advantageous to have the source of a datagram specify the route a datagram takes through the network. This process is called *source routing*.

There are two IP options that support source routing. In each, the option includes a list of IP addresses that specify the routers that must be used to reach the destination. When *strict* source routing is used, the path specified in the option must be used exactly, in sequence, with no other routers permitted to handle the datagram at all. In contrast, *loose* source routing specifies a list of IP addresses that must be followed in sequence, but it allows intervening hops between the devices on the list. (For full details on the exact structure used by each option type, please refer to RFC 791.)

# 22

## **IP DATAGRAM SIZE, FRAGMENTATION, AND REASSEMBLY**



The main responsibility of the Internet Protocol (IP) is to deliver data between internetworked devices. As you saw in the preceding chapter, this requires that data received from higher layers be encapsulated into IP datagrams for transmission. These datagrams are then passed down to the data link layer, where they are sent over physical network links. In order for this to work properly, each datagram must be small enough to fit within the frame format of the underlying technology. If the message is bigger than the maximum frame size of the underlying network, it may be necessary to fragment the message. The datagrams are then sent individually and reassembled into the original message.

IP is designed to manage datagram size and to make fragmentation and reassembly seamless. This chapter explores issues related to managing the size of IP datagrams. I start with an overview of datagram size issues and the important concept of a network's maximum transmission unit (MTU),

discussing why fragmentation is necessary. I then describe the process by which messages are fragmented by the source device, and possibly by routers along the path to the destination, and how they are reassembled by the recipient.

**BACKGROUND INFORMATION** *Understanding fragmentation and reassembly requires some knowledge of the basic format of IP datagrams and some of the fields they contain. If you haven't yet read the chapter describing the general format of IP datagrams in Chapter 21, you may wish to review it before proceeding here.*

## IP Datagram Size, MTU, and Fragmentation Overview

As the core network layer protocol of the TCP/IP protocol suite, IP is designed to implement potentially large internetworks of devices. When we work with IP, we get used to the concept of hosts being able to send information back and forth, even though the hosts may be quite far apart. Although we can usually consider the TCP/IP internetwork to be like a large, abstract virtual network of devices, we must always remember that underneath the network layer, data always travels across one or more physical networks. The implementation of IP must take this reality into account as well.

In order to send messages using IP, we encapsulate the higher-layer data into IP datagrams. These datagrams must then be sent down to the data link layer, where they are further encapsulated into the frames of whatever technology will be used to physically convey them, either directly to their destination or indirectly to the next intermediate step in their journey to their intended recipient. The data link layer implementation puts the entire IP datagram into the data portion (the payload) of its frame format, just as IP puts transport layer messages—transport headers and all—into its IP Data field. This immediately presents us with a potential issue: matching the size of the IP datagram to the size of the underlying data link layer frame size.

### IP Datagram Size and the Underlying Network Frame Size

The underlying network that a device uses to connect to other devices could be a local area network (LAN) connection (like Ethernet or Token Ring), wireless LAN (WLAN) link (such as 802.11), dial-up connection, Digital Subscriber Line (DSL) connection, T1 link, or other wide area network (WAN) connection. Each physical network will generally use its own frame format, and each format has a limit on how much data can be sent in a single frame. If the IP datagram is too large for the data link layer frame format's payload section, we have a problem!

For example, consider a Fiber Distributed Data Interface (FDDI) network. The maximum size of the data field in FDDI is around 4,470 bytes. This means FDDI can handle an IP datagram of up to 4,470 bytes. In contrast, a regular Ethernet frame uses a frame format that limits the size of the payload it sends to 1,500 bytes. This means that Ethernet cannot deal with IP datagrams greater than 1,500 bytes.

Now, remember that in sending a datagram across an internetwork, it may pass across more than one physical network. To access a site on the Internet, for example, we typically send a request through our local router, which then connects

to other routers that eventually relay the request to the Internet site. Each hop as the datagram is forwarded may use a different physical network, with a different maximum underlying frame size.

The whole idea behind a network layer protocol is to implement this concept of a virtual network where devices can communicate over great distances. This means that higher layers shouldn't need to worry about details like the size limits of underlying data link layer technologies. This task falls to IP.

### **MTU and Datagram Fragmentation**

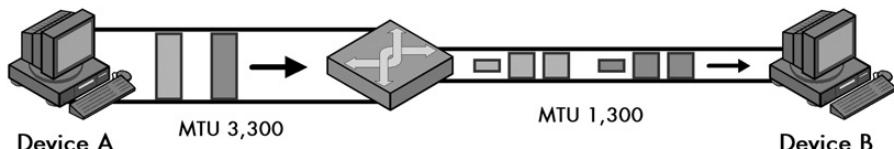
Each device on an IP internetwork must know the capacity of its immediate data link layer connection to other devices. This capacity is called the *maximum transmission unit (MTU)* of the network, also known as the *maximum transfer unit*.

If an IP layer receives a message to be sent across the internetwork, it looks at the size of the message and then computes how large the IP datagram would be after the addition of the 20 or more bytes needed for the IP header. If the total length is greater than the MTU of the underlying network, the IP layer will fragment the message into multiple IP fragments. Thus, if a host is connected to its local network using an Ethernet LAN, it may use an MTU of 1,500 bytes for IP datagrams, and it will fragment anything larger.

Figure 22-1 shows an example of different MTUs and fragmentation.

**KEY CONCEPT** The size of the largest IP datagram that can be transmitted over a physical network is called that network's *maximum transmission unit (MTU)*. If a datagram is passed from a network with a high MTU to one with a low MTU, it must be *fragmented* to fit the other network's smaller MTU.

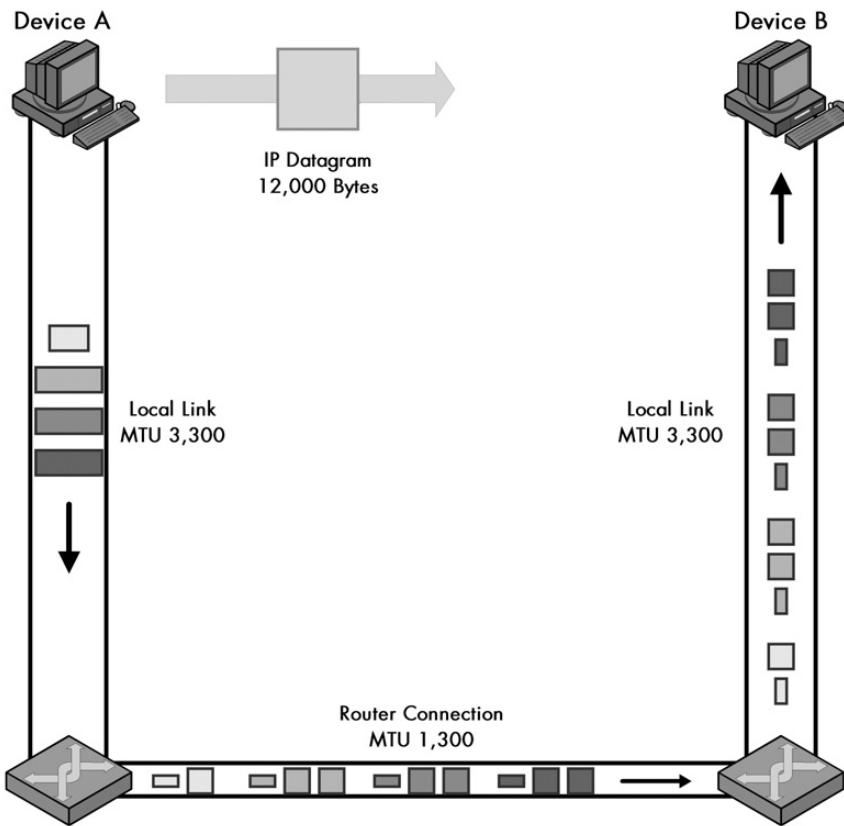
Since some physical networks on the path between devices may have a smaller MTU than others, it may be necessary to fragment the datagram more than once. For example, suppose the source device wants to send an IP message 12,000 bytes long. Its local connection has an MTU of 3,300 bytes. It will need to divide this message into four fragments for transmission: three that are about 3,300 bytes long and a fourth remnant about 2,100 bytes long. (I'm oversimplifying by ignoring the extra headers required; the "The IP Message Fragmentation Process" section later in this chapter includes the full details of the fragmentation process.)



**Figure 22-1: IP maximum transmission unit (MTU) and fragmentation** In this simple example, Device A is sending to Device B over a small internetwork consisting of one router and two physical links. The link from Device A to the router has an MTU of 3,300 bytes, but from the router to Device B, it is only 1,300 bytes. Thus, any IP datagrams larger than 1,300 bytes will need to be fragmented.

## Multiple-Stage Fragmentation

While the IP fragments are in transit, they may need to pass over a hop between two routers where the physical network's MTU is only 1,300 bytes. In this case, each of the fragments will again need to be fragmented. The 3,300-byte fragments will end up in three pieces each (two of about 1,300 bytes and one of around 700 bytes), and the final 2,100-byte fragment will become a 1,300-byte and 800-byte fragment. So, instead of having four fragments, we will end up with eleven ( $3 \times 3 + 1 \times 2$ ) fragments, as shown in Figure 22-2.



**Figure 22-2: IPv4 datagram fragmentation.** This example illustrates a two-step fragmentation of a large IP datagram. The boxes represent datagrams or datagram fragments and are shown to scale. The original datagram is 12,000 bytes, represented by the large, gray box. To transmit this data over the first local link, Device A splits it into four fragments, shown on the left. The first router must fragment each of these into smaller fragments to send them over the 1,300-byte MTU link, as shown on the bottom. Note that the second router does not reassemble the 1,300-byte fragments, even though its link to Device B has an MTU of 3,300 bytes. (The “IP Fragmentation Process” section later in this chapter describes the process by which the fragments in this example are created.)

## **Internet Minimum MTU: 576 Bytes**

Routers are required to handle an MTU of at least 576 bytes. This value is specified in RFC 791; it was chosen to allow a data block of at least 512 bytes, plus room for the standard IP header and options. Since this is the minimum size specified in the IP standard, 576 bytes has become a common default MTU value used for IP datagrams. Even if a host is connected over a local network with an MTU larger than 576 bytes, it may choose to use an MTU value of 576 to ensure that no further fragmentation will be required by intermediate routers.

**NOTE** *While intermediate routers may further fragment an already-fragmented IP message, they do not reassemble fragments. Reassembly is done only by the recipient device. This has some advantages and some disadvantages, as we will see when we examine the reassembly process in the “IP Message Reassembly” section later in this chapter.*

## **MTU Path Discovery**

When we’re trying to send a great deal of data, efficiency in message transmissions becomes important. The larger the IP datagram we send, the smaller the percentage of bytes wasted for overhead such as header fields. This means that, ideally, we want to use the largest MTU possible without requiring fragmentation for its transmission.

To determine the optimal MTU to use for a route between two devices, we would need to know the MTU of every link on that route—information that the endpoints of the connection simply don’t have. However, the connection endpoint can determine the MTU of the overall route by using *MTU path discovery*, which uses an error-reporting mechanism built into TCP/IP Internet Control Message Protocol (ICMP).

One of the message types defined in ICMP version 4 (ICMPv4) is the Destination Unreachable message (see Chapter 32), which is returned under various conditions where an IP datagram cannot be delivered. One of these situations is when a datagram is too large to be forwarded by a router over a physical link, but this datagram has its Don’t Fragment (DF) flag set to prevent fragmentation. In this case, the datagram must be discarded and a Destination Unreachable message sent back to the source. A device can exploit this capability by testing the path with datagrams of different sizes, to see how large they must be before they are rejected.

The source node typically sends a datagram that has the MTU of its local physical link, since that represents an upper bound for any path to or from that device. If this datagram goes through without any errors, the device knows it can use that value for future datagrams to that destination. If it gets back any Destination Unreachable - Fragmentation Needed and DF Set messages, it knows that a link between it and the destination has a smaller MTU. It tries again using a smaller datagram size, and it continues until it finds the largest MTU that can be used on the path.

## IP Message Fragmentation Process

As explained in the previous section, when an IP datagram is too large for the MTU of the underlying data link layer technology used for the next leg of its journey, it must be fragmented before it can be sent across the network. The higher-layer message to be transmitted is not sent in a single IP datagram, but rather broken down into fragments that are sent separately. In some cases, the fragments themselves may need to be fragmented further.

Fragmentation is key to implementing a network-layer internetwork that is independent of lower-layer details, but it introduces significant complexity to IP. Remember that IP is an unreliable, connectionless protocol. IP datagrams can take any of several routes on their way from the source to the destination, and some may not even make it to the destination at all. When a message is fragmented, this converts a single datagram into many, which introduces several new concerns:

**Sequencing and Placement** The fragments will typically be sent in sequential order from the beginning of the message to the end, but they won't necessarily show up in the order in which they were sent. The receiving device must be able to determine the sequence of the fragments to reassemble them in the correct order. In fact, some implementations send the last fragment first, so the receiving device will immediately know the full size of the original, complete datagram. This makes keeping track of the order of segments even more essential.

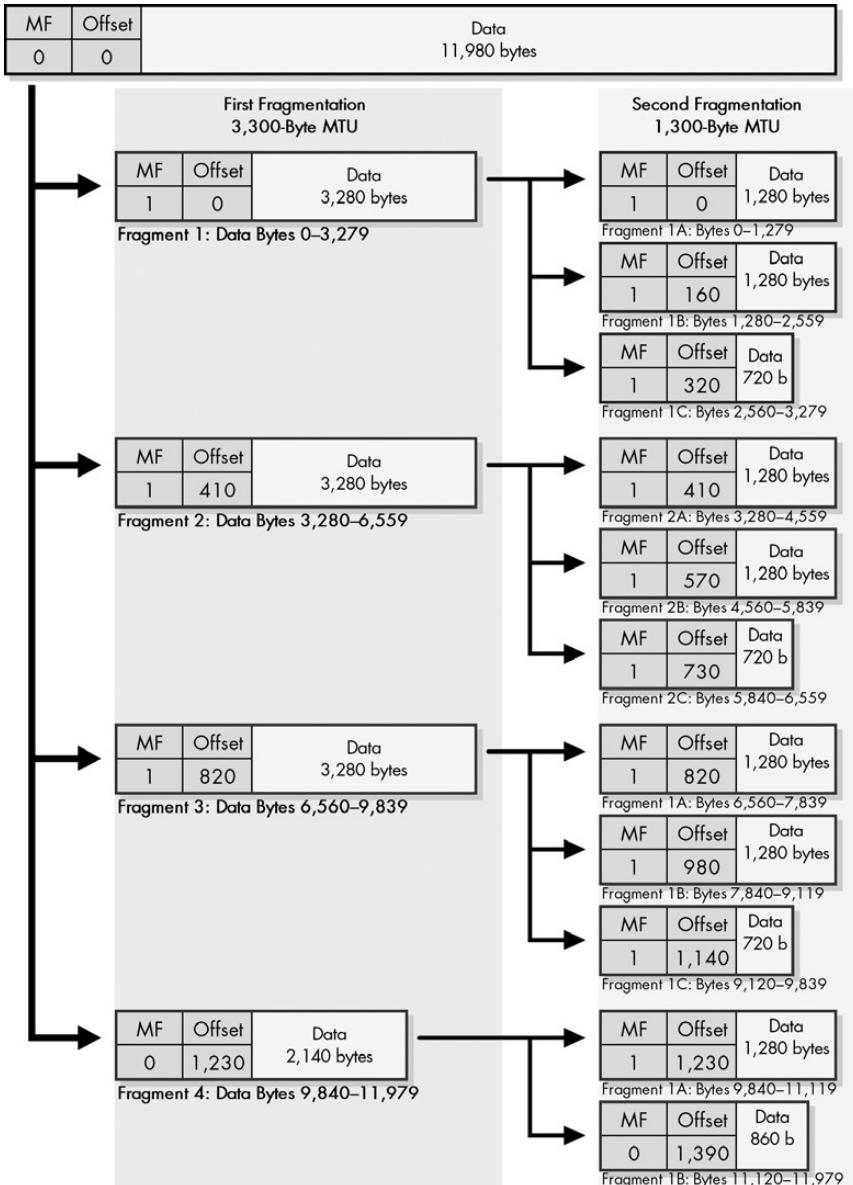
**Separation of Fragmented Messages** A source device may need to send more than one fragmented message at a time, or it may send multiple datagrams that are fragmented en route. This means that the destination may be receiving multiple sets of fragments that must be put back together. Imagine a box containing pieces from two, three, or more jigsaw puzzles, and you understand this issue.

**Completion** The destination device must be able to tell when it has received all of the fragments so it knows when to start reassembly (or when to give up if it didn't get all the pieces).

To address these concerns and allow the proper reassembly of the fragmented message, IP includes several fields in the IP format header that convey information from the source to the destination about the fragments. Some of these fields contain a common value for all the fragments of the message; others are different for each fragment.

### *The IP Fragmentation Process*

The device performing the fragmentation follows a specific algorithm to divide the message into fragments for transmission. The exact implementation of the fragmentation process depends on the device. For example, consider an IP message 12,000 bytes wide (including the 20-byte IP header) that needs to be sent over a link with an MTU of 3,300 bytes. Figure 22-3 depicts a typical method by which this fragmentation might be performed.



**Figure 22-3: IPv4 datagram fragmentation process** In this diagram, the MF and Fragment Offset fields of each fragment are shown for reference. The Data fields are shown to scale (the length of each is proportional to the number of bytes in the fragment).

The four fragments shown in Figure 22-3 are created as follows:

- The first fragment is created by taking the first 3,300 bytes of the 12,000-byte IP datagram. This includes the original header, which becomes the IP header of the first fragment (with certain fields changed, as described in the next section). So, 3,280 bytes of data are in the first fragment. This leaves 8,700 bytes (11,980–3,280) to encapsulate.

- The next 3,280 bytes of data are taken from the 8,700 bytes that remain after the first fragment is built and paired with a new header to create the second fragment. This leaves 5,420 bytes.
- The third fragment is created from the next 3,280 bytes of data, with a 20-byte header. This leaves 2,140 bytes of data.
- The remaining 2,140 bytes are placed into the fourth fragment, with a 20-byte header.

There are two important points here. First, IP fragmentation does *not* work by fully encapsulating the original IP message into the Data fields of the fragments. If this were the case, the first 20 bytes of the Data field of the first fragment would contain the original IP header. (This technique is used by some other protocols, such as the PPP Multilink Protocol, discussed in Chapter 9.) The original IP header is transformed into the IP header of the first fragment.

Second, note that the total number of bytes transmitted increases: we are sending 12,060 bytes ( $3,300*3+2,160$ ), instead of 12,000 bytes. The extra 60 bytes are from the additional headers in the second, third, and fourth fragments. (The increase in size could theoretically be even larger if the headers contain options.)

### ***Fragmentation-Related IP Datagram Header Fields***

When a sending device or router fragments a datagram, it must provide information that will allow the receiving device to identify the fragments and reassemble them into the original datagram. This information is recorded by the fragmenting device in a number of fields in the IP datagram header:

**Total Length** After fragmenting, the Total Length field indicates the length of each fragment, not the length of the overall message. Normally, the fragment size is selected to match the MTU value in bytes. However, fragments must have a length that is a multiple of 8, to allow proper offset specification (handled by the Fragment Offset field). The last fragment will usually be shorter than the others because it will contain a leftover piece, unless the message length happens to be an exact multiple of the fragment size.

**Identification** To solve the problem of pieces from many jigsaw puzzles in the same box, a unique identifier is assigned to each message being fragmented. This is like writing a different number on the bottom of each piece of a jigsaw puzzle before tossing it in the box. This value is placed in the Identification field in the IP header of each fragment sent. The Identification field is 16 bits wide, so a total of 65,536 different identifiers can be used. Obviously, we want to make sure that each message that is being fragmented for delivery has a different identifier. The source can decide how it generates unique identifiers. This may be done through something as simple as a counter that is incremented each time a new set of fragments is created.

**More Fragments** The More Fragments flag is set to a 1 for all fragments except the last one, which has it set to 0. When the fragment with a value of 0 in the More Fragments flag is seen, the destination knows it has received the last fragment of the message.

**Fragment Offset** The Fragment Offset field solves the problem of sequencing fragments by indicating to the recipient device where in the overall message each particular fragment should be placed. The field is 13 bits wide, so the offset can be from 0 to 8,191. Fragments are specified in units of 8 bytes, which is why fragment length must be a multiple of 8. Uncoincidentally,  $8,191 \times 8$  is 65,528, just about the maximum size allowed for an IP datagram. In the example shown in Figure 22-3, the first fragment would have a Fragment Offset of 0, the second would have an offset of 410 ( $3,280/8$ ), the third would have an offset of 820 ( $6,560/8$ ), and the fourth would have an offset of 1,230.

An IP datagram has a couple of other fields related to fragmentation. First, if a datagram containing options must be fragmented, some of the options may be copied to each of the fragments. This is controlled by the Copied flag in each option field.

Second, in the IP header, there is a flag called Don't Fragment. This field can be set to 1 by a transmitting device to specify that a datagram should not be fragmented in transit. This may be used in certain circumstances where the entire message must be delivered intact for some reason. It may also be used if the destination device has a limited IP implementation and cannot reassemble fragments, and it is also used for testing the MTU of a link. Normally, however, devices don't care about fragmentation, and this field is left at 0.

If a router encounters a datagram too large to pass over the next physical network but with the Don't Fragment bit set to 1, it cannot fragment the datagram and it cannot pass it along either, so it is stuck. It will generally drop the datagram and return an ICMP Destination Unreachable error message: “Fragmentation Needed and Don't Fragment Bit Set.” This is used in MTU path discovery, as described earlier in this chapter.

**KEY CONCEPT** When an MTU requirement forces a datagram to be fragmented, it is split into several smaller IP datagrams, each containing part of the original. The header of the original datagram is changed into the header of the first fragment, and new headers are created for the other fragments. Each is set to the same Identification value to mark them as part of the same original datagram. The Fragment Offset of each is set to the location where the fragment belongs in the original. The More Fragments field is set to 1 for all fragments but the last, to let the recipient know when it has received all the fragments.

## IP Message Reassembly

When a datagram is fragmented, it becomes multiple fragment datagrams. The destination of the overall message must collect these fragments and reassemble them into the original message.

While reassembly is the complement to fragmentation, the two processes are not symmetric. A primary differentiation between the two is that intermediate routers can fragment a single datagram or further fragment a datagram that is already a fragment, but intermediate devices do not perform reassembly; reassembly happens only at the message's ultimate destination. Thus, if a datagram at an intermediate router on one side of a physical network with an MTU of 1,300 bytes causes fragmentation of

a 3,300-byte datagram, the router on the other end of this 1,300 MTU link will *not* restore the 3,300-byte datagram to its original state. It will send all the 1,300-byte fragments on down the internetwork, as shown in Figure 22-2, earlier in the chapter.

In IP version 4 (IPv4), fragmentation can be performed by a router between the source and destination of an IP datagram, but reassembly is done only by the destination device.

There are a number of reasons why the decision was made to implement IP reassembly this way. Perhaps the most important reason is that fragments can take different routes to get from the source to destination, so any given router may not see all the fragments in a message. Another reason is that if routers needed to worry about reassembling fragments, their complexity would increase. Finally, reassembly of a message requires that we wait for all fragments before sending on the reassembled message. Having routers do this would slow down routing. Since routers don't reassemble messages, they can immediately forward all fragments on to the ultimate recipient.

However, there are drawbacks to this design as well. One is that it results in more, smaller fragments traveling over longer routes than if intermediate reassembly occurred. This increases the chances of a fragment getting lost and the entire message being discarded. Another is a potential inefficiency in the utilization of data link layer frame capacity. In the example of a 3,300-byte datagram being fragmented for a 1,300-byte MTU link, the 1,300-byte fragments would not be reassembled back into a 3,300-byte datagram at the end of the 1,300-MTU link. If the next link after that one also had an MTU of 3,300 bytes, we would need to send three frames, each encapsulating a 1,300-byte fragment, instead of a single larger frame, which is slightly slower.

As described in the previous section, several IP header fields are filled in when a message is fragmented to give the receiving device the information it requires to properly reassemble the fragments. The receiving device follows a procedure to keep track of the fragments as they are received and build up its copy of the total received message from the source device. Most of its efforts are geared toward dealing with the potential difficulties associated with IP being an unreliable protocol.

The details of implementation of the reassembly process are specific to each device, but reassembly generally includes the following functions:

**Fragment Recognition and Fragmented Message Identification** The recipient knows it has received a message fragment the first time it sees a datagram with the More Fragments bit set to 1 or the Fragment Offset a value other than 0. It identifies the message based on the source and destination IP addresses, the protocol specified in the header, and the Identification field generated by the sender.

**Buffer Initialization** The receiving device initializes a buffer where it can store the fragments of the message as they are received. It keeps track of which portions of this buffer have been filled with received fragments, perhaps using a special table. By doing this, it knows when the buffer is partially filled with received fragments and when it is completely full.

**Timer Initialization** The receiving device sets up a timer for reassembly of the message. Since it is possible that some fragments may never show up, this timer ensures that the device will not wait an infinite time trying to reassemble the message.

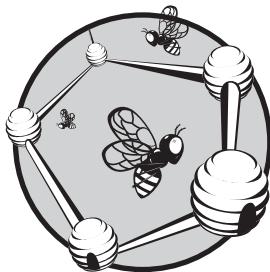
**Fragment Receipt and Processing** Whenever a fragment of this message arrives (as indicated by its having the same source and destination addresses, protocol, and Identification as the first fragment), the fragment is processed. It is inserted into the message buffer in the location indicated by its Fragment Offset field. The device also makes note of the fact that this portion of the message has been received.

Reassembly is complete when the entire buffer has been filled and the fragment with the More Fragments bit set to 0 is received, indicating that it is the last fragment of the datagram. The reassembled datagram is then processed in the same way as a normal, unfragmented datagram. On the other hand, if the timer for the reassembly expires with any of the fragments missing, the message cannot be reconstructed. The fragments are discarded, and an ICMP Time Exceeded message is generated. Since IP is unreliable, it relies on higher-layer protocols such as the Transmission Control Protocol (TCP) to determine that the message was not properly received and then retransmit it.



# 23

## IP ROUTING AND MULTICASTING



The essential functions of Internet Protocol (IP) datagram encapsulation and addressing are sometimes compared to putting a letter in an envelope and then writing the address of the recipient on it. Once our IP datagram “envelope” is filled and labeled, it is ready to go, but it’s still sitting on our desk. The last of the main functions of IP is to get the envelope to our intended recipient. This is the process of datagram *delivery*. When the recipient is not on our local network, this delivery requires that the datagram be *routed* from our network to the one where the destination resides.

This chapter concludes our look at IP version 4 (IPv4) with a discussion of some of the particulars of how it routes datagrams over an internetwork. I begin with an overview of the process and contrast direct and indirect delivery of data between devices. I discuss the main method used to route

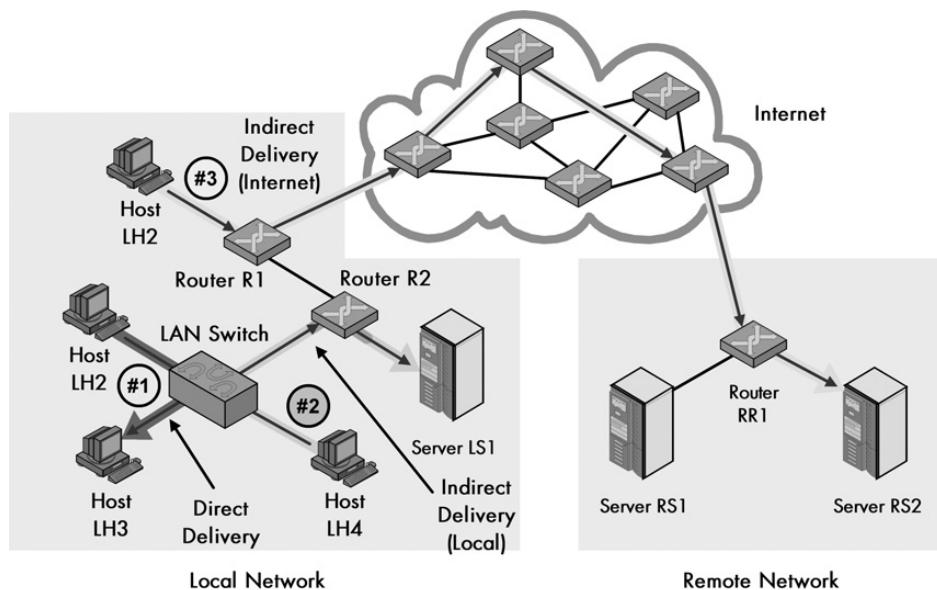
datagrams over the internetwork, and I explain briefly how IP routing tables are built and maintained. I describe how the move from classful to classless addressing using Classless Inter-Domain Routing (CIDR) has impacted routing.

I conclude with a brief look at the issues related to IP multicasting. Multicasting isn't really a part of routing, but many of the issues in multicasting are related to datagram delivery and routing.

**RELATED INFORMATION** This chapter focuses on routing issues that are directly related to how IP works. Routing is a complex and important topic in networking, and you'll find much more information about it in Chapters 37 through 41.

## IP Datagram Delivery

The overall job of IP is to transmit messages from higher-layer protocols over an internetwork of devices. These messages must be packaged and addressed, and fragmented if necessary, and then they must be *delivered*. The process of delivery can be either simple or complex, depending on the proximity of the source and destination devices. We can divide all IP datagram deliveries into two general types: direct delivery and indirect delivery. Figure 23-1 shows some examples of IP datagram delivery types.



**Figure 23-1: Direct and indirect (routed) delivery of IP datagrams** This diagram shows three examples of IP datagram delivery. The first transmission (#1, dark arrow) shows a direct delivery between two devices on the local network. The second (#2, light arrow) shows indirect delivery within the local network, between a client and server separated by a router. The third (#3, medium arrow) shows a more distant indirect delivery, between a client on the local network and a server across the Internet.

## **Direct Datagram Delivery**

When datagrams are sent between two devices on the same physical network, the datagrams may be delivered directly from the source to the destination. For example, if you wanted to deliver a letter to a neighbor on your street, you would probably just put her name on the envelope and stick it right in her mailbox.

Direct delivery is obviously a simple delivery method. The source simply sends the IP datagram down to its data link layer implementation. The data link layer encapsulates the datagram in a frame that is sent over the physical network directly to the recipient's data link layer, which passes it up to the IP layer.

## **Indirect Datagram Delivery (Routing)**

When two devices are not on the same physical network, the delivery of datagrams from one to the other is *indirect*. Since the source device cannot see the destination on its local network, it must send the datagram through one or more intermediate devices to deliver it. Indirect delivery is like mailing a letter to a friend in a different city. You don't deliver it yourself; you use the postal system. The letter journeys through the postal system, possibly taking several intermediate steps, and ends up in your friend's neighborhood, where a postal carrier puts it into his mailbox.

Indirect delivery is much more complicated, because we can't send the data straight to the recipient. In fact, we usually will not even know exactly where the recipient is. Sure, we have its address, but we may not know what network it is on, or where that network is relative to our own. (If I told you my address, you would know it's somewhere in Bennington, Vermont, but could you find it?) Just as we must rely on the postal system in the envelope analogy, we must rely on the internetwork itself to indirectly deliver datagrams. And like the postal system, IP doesn't require you to know how to get the message to its recipient; you just put it into the system.

The devices that accomplish this magic of indirect delivery are generally known as *routers*, and indirect delivery is more commonly called *routing*. Like entrusting a letter to your local mail carrier or mailbox, a host that needs to deliver a message to a distant device generally sends datagrams to its local router. The router connects to one or more other routers, and they each maintain information about where to send datagrams so that they reach their final destination.

Indirect delivery is almost always required when communicating with distant devices, such as those on the Internet or across a wide area network (WAN) link. However, it may also be needed even to send a message to a device in the next room of your office, if that device is not connected directly to your device at layer 2.

**NOTE** *In the past, routers were often called gateways. Today, this term more generally can refer to devices that connect networks in a variety of ways. You will still sometimes hear routers called gateways, especially in the context of terms like default gateway, but since it is ambiguous, the term router is preferred.*

## **The Relationship Between Datagram Routing and Addressing**

Each time a datagram is to be sent, the sender must determine first whether it can be delivered directly or if routing is required. IP addressing is what allows a device to quickly determine whether or not it is on the same network as its intended recipient. The following are the three main categories of addressing (see Chapter 16):

**Conventional Classful Addressing** We know the class of each address by looking at the first few bits. This tells us which bits of an address are the network ID. If the network ID of the destination is the same as our own, the recipient is on the same network; otherwise, it is not. Refer to Chapter 17 for more on classful addressing.

**Subnetted Classful Addressing** We use our subnet mask to determine our network ID and subnet ID and that of the destination address. If the network ID and subnet are the same, the recipient is on the same subnet. If only the network ID is the same, the recipient is on a different subnet of the same network. If the network ID is different, the destination is on a different network entirely. See Chapter 18 for a full discussion of subnetting.

**Classless Addressing** The same basic technique is used as for subnetted classful addressing, except that there are no subnets. We use the slash number to determine what part of the address is the network ID and compare the source and destination as before; see Chapter 20. (There are complications here, however, as discussed in the “IP Routing in a Subnet or Classless Addressing (CIDR) Environment” section later in this chapter.)

**KEY CONCEPT** The delivery of IP datagrams is divided into two categories: *direct* and *indirect*. Direct delivery is possible when two devices are on the same physical network. When they are not, indirect delivery, more commonly called *routing*, is required to get the datagrams from the source to the destination. A device can tell which type of delivery is required by looking at the IP address of the destination, in conjunction with supplemental information such as the subnet mask, which tells the device what network or subnet it is on.

The determination of what type of delivery is required is the first step in the source deciding where to send a datagram. If it realizes the destination is on the same local network, it will address the datagram to the recipient directly at the data link layer. Otherwise, it will send the datagram to the data link layer address of one of the routers to which it is connected. The IP address of the datagram will still be that of the ultimate destination. Mapping between IP addresses and data link layer addresses is accomplished using the TCP/IP Address Resolution Protocol (ARP), which is discussed in Chapter 13.

Routing is done in indirect delivery to get the datagram to the local network of the recipient. Once the datagram has been routed to the recipient’s physical network, it is sent to the recipient by the recipient’s local router. So, you could say that indirect delivery includes direct delivery as its final step.

**NOTE** Strictly speaking, any process of delivery between a source and destination device can be considered routing, even if the devices are on the same network. It is common, however, for the process of routing to refer more specifically to indirect delivery.

## IP Routing Concepts and the Process of Next-Hop Routing

IP's ability to route information is what allows us to use it to create the equivalent of a virtual internetwork that spans potentially thousands of physical networks, allowing devices even on opposite ends of the globe communicate. Let's take a brief look at key IP routing concepts.

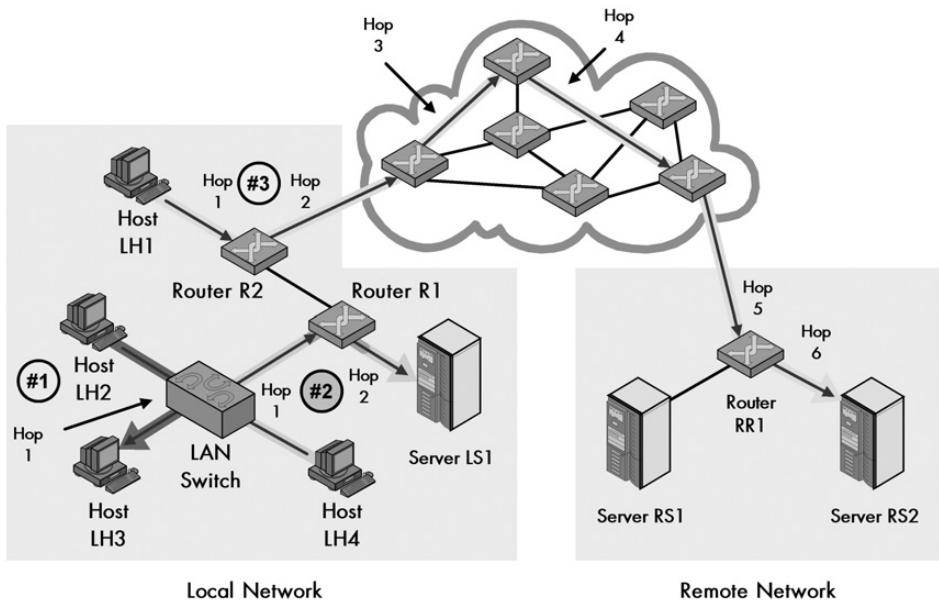
To continue with our postal system analogy, I can send a letter from my home in the United States to someone in, say, India, and the postal systems of both countries will work (or should work) to deliver the letter to its destination. However, when I drop a letter in the mailbox, it's not like someone shows up, grabs the letter, and hand-delivers it to the right address in India. The letter travels from the mailbox to my local post office. From there, it probably goes to a regional distribution center, and then from there, to a hub for international traffic. It goes to India, perhaps via an intermediate country. When it gets to India, the Indian postal system uses its own network of offices and facilities to route the letter to its destination. The envelope hops from one location to the next, until it reaches its destination.

IP routing works in very much the same manner. Even though IP lets devices connect over the internetwork using indirect delivery, all of the actual communication of datagrams occurs over physical networks using routers. We don't know exactly where the destination device's network is, and we certainly don't have any way to connect directly to each of the thousands of networks out there. Instead, we rely on these intermediate devices—routers—that are each physically connected to each other in a variety of ways to form a mesh containing millions of paths between networks. The datagram is handed off from one router to the next, until it gets to the physical network of the destination device. This process is called *next-hop routing*, as illustrated in Figure 23-2.

This is a critical concept in how IP works: routing is done step by step, one hop at a time. When we decide to send a datagram to a device on a distant network, we don't know the exact path that the datagram will take; we have only enough information to send it to the correct router to which we are attached. That router, in turn, looks at the IP address of the destination and decides where the datagram should hop to next. This process continues until the datagram reaches the destination host's network.

At first, next-hop routing may seem like a strange way of sending datagrams over an internetwork. In fact, it is part of what makes IP so powerful. On each step of the journey to any other host, a router needs to know only where the next step for the datagram is. Without this concept, each device and router would need to know what path to take to every other host on the internetwork, which would be quite impractical.

**KEY CONCEPT** Indirect delivery of IP datagrams is accomplished using a process called *next-hop routing*, where each message is handed from one router to the next until it reaches the network of the destination. The main advantage of this is that each router needs to know only which neighboring router should be the next recipient of a given datagram, rather than needing to know the exact route to every destination network.



**Figure 23-2: IP datagram next-hop routing** This is the same diagram as that shown in Figure 23-1, except it explicitly shows the hops taken by each of the three sample transmissions. The direct delivery of the first transmission has only one hop (remember that the switch doesn't count because it is invisible at layer 3). The local indirect delivery passes through one router, so it has two hops. The Internet delivery has six hops. (Actual Internet routes can be much longer.)

Another key concept related to the principle of next-hop routing is that routers, not hosts, are designed to accomplish routing. Most hosts are connected to the rest of the internetwork (or Internet) using only one router. It would be a maintenance nightmare to need to give each host the intelligence to know how to route to every other host. Instead, hosts decide only if they are sending to their own local network or to another network. If the destination is another network, a host just sends the datagram to its router and says, “Here, you take care of this.” If a host has a connection to more than one router, it needs to know only which router to use for certain sets of distant networks.

Again, each hop consists of the traversal of a physical network. Once a source sends a datagram to its local router, the data link layer on the router passes it up to the router’s IP layer. There, the datagram’s header is examined, and the router decides which device should get the datagram next. It then passes the datagram back down to the data link layer to be sent over one of the router’s physical network links, typically to another router. The router will have a record of the physical addresses of the routers to which it is connected, or it will use ARP to determine these addresses.

## IP Routes and Routing Tables

As described in the previous section, routers are responsible for forwarding traffic on an IP internetwork. Each router accepts datagrams from a variety of sources, examines the IP address of the destination, and decides the next hop that the datagram needs to take to get it that much closer to its final destination. But how does a router know where to send different datagrams?

Each router maintains a set of information that provides a mapping between different network IDs and the other routers to which it is connected. This information is contained in a data structure normally called a *routing table*. Each entry in the table, called a *routing entry*, provides information about one network (or subnetwork or host). It basically says, “If the destination of this datagram is in the following network, the next hop you should take is to the following device.” Each time a datagram is received, the router checks its destination IP address against the routing entries in its table to decide where to send the datagram and then sends it on to its next hop.

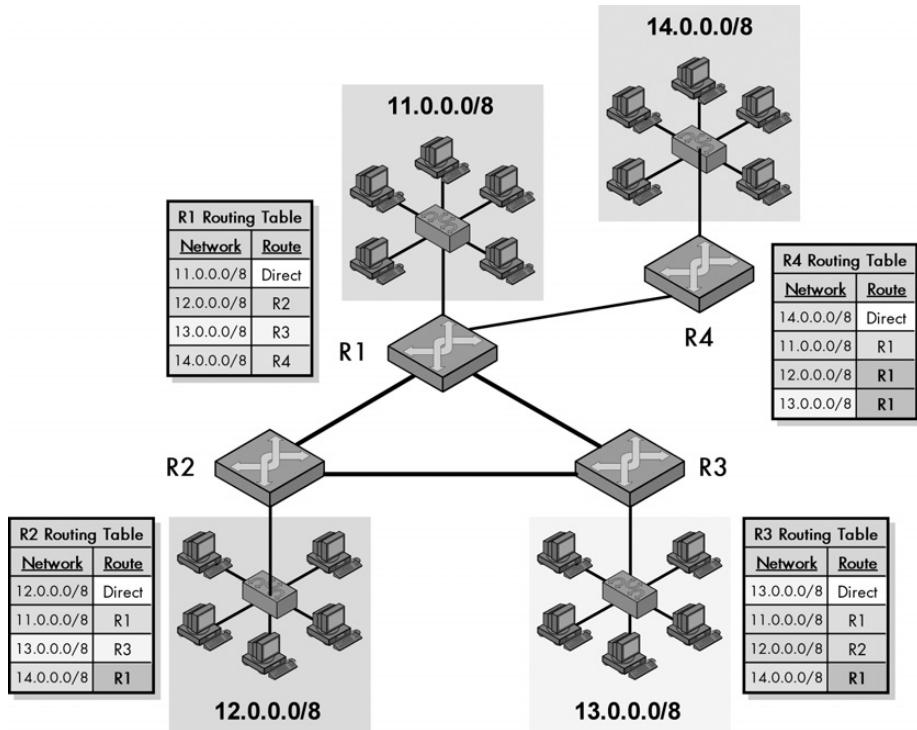
**KEY CONCEPT** A router make decisions about how to route datagrams using its internal *routing table*. The table contains entries specifying to which router datagrams should be sent in order to reach a particular network.

Obviously, the fewer entries in this table, the faster the router can decide what to do with datagrams. (This was a big part of the motivation for classless addressing, which aggregates routes into supernetworks to reduce router table size, as described in the next section.) Some routers have connections to only two other devices, so they don’t have much of a decision to make. Typically, the router will simply take datagrams coming from one of its interfaces and, if necessary, send them out on the other one. For example, consider a small company’s router acting as the interface between a network of three hosts and the Internet. Any datagrams sent to the router from a host on this network will need to go over the router’s connection to the router at the Internet service provider (ISP).

When a router has connections to more than two devices, things become considerably more complex. A certain distant network may be more easily reachable using a particular connection. The routing table not only contains information about the networks directly connected to the router, but also information that the router has learned about more distant networks.

Figure 23-3 shows an example with four routers. Routers R1, R2, and R3 are connected in a triangle, so that each router can send directly to the others, as well as to its own local network. R1’s local network is 11.0.0.0/8, R2’s is 12.0.0.0/8, and R3’s is 13.0.0.0/8. R1 knows that any datagram it sees with 11 as the first octet is on its local network. It will also have a routing entry that says that any IP address starting with 12 should go to R2, and any IP address starting with 13 should go to R3. R1 also connects to router R4, which has 14.0.0.0/8 as its local network. R1 will have an entry for this local network, but R2 and R3 also need to know how to reach 14.0.0.0/8, even though they don’t connect to its router directly. Most likely, they

will have an entry that says that any datagrams intended for 14.0.0.0/8 should be sent to R1. R1 will then forward them to R4. Similarly, R4 will send any traffic intended for 12.0.0.0/8 or 13.0.0.0/8 through R1.



**Figure 23-3: IP routing and routing tables** This diagram shows a small, simple internetwork consisting of four LANs each served by a router. The routing table for each lists the router to which datagrams for each destination network should be sent. Notice that due to the triangle, R1, R2, and R3 can send to each other. However, R2 and R3 must send through R1 to deliver to R4, and R4 must use R1 to reach either of the others.

Now, imagine that this process is expanded to handle thousands of networks and routers. Not only do routers need to know which of their local connections to use for each network, but they want to know, if possible, what is the *best* connection to use for each network. Since routers are interconnected in a mesh, there are usually multiple routes between any two devices, but we want to take the best route whenever we can. This may be the shortest route, the least congested route, or the route considered optimal based on other criteria.

Determining which routes we should use for different networks is an important but very complex job. Routers plan routes and exchange information about routes and networks using IP *routing protocols*. R2 and R3 use these protocols to find out that 14.0.0.0/8 exists and that it is connected to them via R1. (I discuss these support protocols in Chapters 37 through 41.)

**NOTE** There is a difference between a routable protocol and a routing protocol. IP is a routable protocol, which means its messages (datagrams) can be routed. Examples of routing protocols are the Routing Information Protocol (RIP) and Border Gateway Protocol (BGP), which are used to exchange routing information between routers (see Chapter 38 and Chapter 40).

## IP Routing in a Subnet or Classless Addressing (CIDR) Environment

As discussed in the previous chapters, there are three main categories of IP addressing: classful, subnetted classful, and classless. The method used for determining whether direct or indirect delivery of a datagram is required is different for each type of addressing. The type of addressing used in the network also impacts how routers decide to forward traffic in an internetwork.

One of the main reasons why the traditional class-based addressing scheme was created was that it made both addressing and routing relatively simple. Remember that IPv4 was developed in the late 1970s, when the cheap and powerful computer hardware we take for granted today was still science fiction. For the internetwork to function properly, routers needed to be able to look at an IP address and quickly decide what to do with it.

Classful addressing was intended to make this possible. There was only a two-level hierarchy for the entire internetwork: network ID and host ID. Routers could tell by looking at the first four bits which of the bits in any IP address were the network ID and which were the host ID. Then they needed only consult their routing tables to find the network ID and see which router offered the best route to that network.

The addition of subnetting to conventional addressing didn't really change this for the main routers on the Internet, because subnetting is internal to the organization. The main routers handling large volumes of traffic on the Internet didn't look at subnets at all. The additional level of hierarchy that subnets represent existed only for the routers within each organization that chose to use subnetting. These routers, when deciding what to do with datagrams within the organization's network, needed to extract not only the network ID of IP addresses, but also the subnet ID. This told them which internal physical network should get the datagram.

Classless addressing is formally called *Classless Inter-Domain Routing (CIDR)*. The fact that the name includes *routing* but not *addressing* is evidence that CIDR was introduced in large part to improve the efficiency of routing. This improvement occurs because classless networks use a multiple-level hierarchy. Each network can be broken down into subnetworks, sub-subnetworks, and so on. This means that when we are deciding how to route in a CIDR environment, we can also describe routes in a hierarchical manner. Many smaller networks can be described using a single, higher-level network description that represents them all to routers in the rest of the internetwork. This technique, sometimes called *route aggregation*, reduces routing table size.

Let's refer back to the detailed example I presented in Chapter 20. An ISP started with the block 71.94.0.0/15 and subdivided it multiple times to create smaller blocks for itself and its customers. To the customers and users of this block, these smaller blocks must be differentiated; the ISP obviously needs to know how to route traffic to the correct customer. To everyone else on the Internet, however, these details are unimportant in deciding how to route datagrams to anyone in that ISP's block.

For example, suppose I am using a host with IP address 211.42.113.5 and I need to send a message to 71.94.1.43. My local router and the main routers on the Internet don't know where in the 71.94.0.0/15 block that address is, and they don't need to know. They just know that anything with the first 15 bits containing the binary equivalent of 71.94 goes to the router that handles 71.94.0.0/15, which is the aggregated address of the entire block. They let the ISP's routers figure out which of its constituent subnetworks contains 71.94.1.43.

Contrast this with a classful environment. Here, each of the customers of this ISP would probably have one or more Class C address blocks, each of which would require a separate routing entry, and these blocks would need to be known by *all* routers on the Internet. Thus, instead of just one 71.94.0.0/15 entry, there would be dozens or even hundreds of entries for each customer network. In the classless scheme, only one entry exists, for the parent ISP.

CIDR provides benefits to routing but also increases its complexity. Under CIDR, we cannot determine which bits are the network ID and which are the host ID just from the IP address. To make matters worse, we can have networks, subnetworks, sub-subnetworks, and so on that all have the same base address!

In our example, 71.94.0.0/15 is the complete network, and subnetwork 0 is 71.94.0.0/16. They have a different prefix length (the number of network ID bits) but the same base address. If a router has more than one match for a network ID in this manner, it must use the match with the longest network identifier first, since it represents a more specific network description.

## IP Multicasting

The great bulk of TCP/IP communications uses IP to send messages from one source device to one recipient device, in a process called *unicast* communication. This is the type of messaging we normally use TCP/IP for, so when you use the Internet, you are using unicast for pretty much everything.

IP does, however, also support the ability to have one device send a message to a set of recipients. This is called *multicasting*. IP multicasting has been officially supported since IPv4 was first defined, but has not seen widespread use over the years, due largely to lack of support for multicasting in many hardware devices. Interest in multicasting has increased in recent years, and support for multicasting was made a standard part of the next-generation IP version 6 (IPv6) protocol. Here, we will take a brief look at multicasting, which is a large and complex subject.

The idea behind IP multicasting is to allow a device on an IP internetwork to send datagrams not to just one recipient, but to an arbitrary collection of other devices. IP multicasting is modeled after the similar function used in the data link layer to allow a single hardware device to send to various members of a group. Multicasting is relatively easy at the data link layer, however, because all the devices can communicate directly. In contrast, at the network layer, we are connecting together devices that may be quite far away from each other and must route datagrams between these different networks. This necessarily complicates multicasting when done using IP (except in the special case where we use IP multicasting only between devices on the same data link layer network).

There are three primary functions that must be performed to implement IP multicasting: addressing, group management, and datagram processing/routing.

## **Multicast Addressing**

Special addressing must be used for multicasting. A *multicast address* identifies not a single device, but a *multicast group* of devices that listen for certain datagrams sent to them. In IPv4, one-sixteenth of the entire address space was set aside for multicast addresses: the Class D block of the original classful addressing scheme. Various techniques are used to define the meaning of addresses within this block and to define a mapping between IP multicast and data link layer multicast addresses. (See the discussion of IP multicast addressing in Chapter 17; mapping of IP multicast addresses to hardware layer multicast addresses is discussed in Chapter 13.)

## **Multicast Group Management**

Group management encompasses all of the activities required to set up groups of devices. Devices must be able to dynamically join groups and leave groups, and information about groups must be propagated around the IP internetwork. To support these activities, additional techniques are required. The *Internet Group Management Protocol (IGMP)* is the chief tool used for this purpose. It defines a message format to allow information about groups and group membership to be sent between devices and routers on the Internet.

## **Multicast Datagram Processing and Routing**

Handling and routing datagrams in a multicast environment is probably the most complicated function. There are several issues here:

- Since we are sending from one device to many, we need to actually create multiple copies of the datagram for delivery, in contrast to the single datagram used in the unicast case. Routers must be able to tell when they need to create these copies.
- Routers must use special algorithms to determine how to forward multicast datagrams. Since each one can lead to many copies being sent to various places, efficiency is important to avoid creating unnecessary volumes of traffic.
- Routers must be able to handle datagrams sent to a multicast group, even if the source is not a group member.

Routing in a multicast environment requires significantly more intelligence on the part of router hardware. Several special protocols, such as the Distance Vector Multicast Routing Protocol (DVMRP) and the multicast version of Open Shortest First (OSPF), are used to enable routers to forward multicast traffic effectively. These algorithms must balance the need to ensure that every device in a group receives a copy of all datagrams intended for that group with the need to prevent unnecessary traffic from moving across the internetwork.

**KEY CONCEPT** IP multicasting allows special applications to be developed where one device sends information to multiple devices, across a private internetwork or the global Internet. It is more complex than conventional unicast IP and requires special attention, particularly in the areas of addressing and routing.

This overview has only scratched the surface of IP multicasting. The complexity involved in handling groups and forwarding messages to multicast groups is one reason why support for the feature has been quite uneven and, as a consequence, it is not used widely. Another issue is the demanding nature of multicasting: It uses a great deal of network bandwidth for copies of messages, and it also requires more work of already-busy routers.

# PART II-4

## INTERNET PROTOCOL VERSION 6 (IPV6)

Since 1981, TCP/IP has been built on version 4 of the Internet Protocol (IPv4), discussed at length in the preceding part. IPv4 was created when the giant, worldwide Internet we take for granted today was just a small, experimental network. Considering how much the Internet has grown and changed over the course of two decades, IPv4 has done its job admirably. At the same time, it has been apparent for many years that certain limitations in this venerable protocol would hold back the future growth of the Internet if they were not addressed.

Due to the key role that IP plays, changing it is no simple feat. It means a substantial modification to the way that nearly everything in TCP/IP operates. However, even though we find change difficult, most of us know that it is necessary. For the past several years, development of a new version of IP has been under way, officially called *Internet Protocol version 6 (IPv6)* and also sometimes referred to as *IP Next Generation* or *IPng*. IPv6 is poised to take over for IPv4, and it will be the basis for the Internet of the future.

In this part, I provide a detailed description of IPv6. Since IPv6 is still IP, just like IPv4, it performs the same functions: addressing, encapsulation, fragmentation and reassembly, and datagram delivery and routing. For this reason, this discussion of IPv6 is patterned after the discussion of IPv4. There

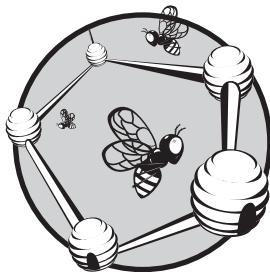
are four chapters: The first covers IPv6 concepts and issues; the second discusses IPv6 addressing; the third discusses IPv6 encapsulation and formatting; and the fourth discusses IPv6 datagram fragmentation, reassembly, and routing.

Since IPv6 represents the evolution of IP, many of its concepts of operation are built on those introduced in IPv4. To avoid unnecessary duplication in this part, I've assumed you are familiar with the operation of IPv4, especially addressing and how datagrams are packaged and delivered. If you have not read Part II-3, reviewing it first would be wise, because the description of IPv6 focuses on how it differs from the current IP version.

You may also wish to refer to the Part II-6, which covers the Internet Control Message Protocol (ICMP), part of which is ICMP version 6—ICMP for IPv6, and the IPv6 Neighbor Discovery (ND) protocol, since these are companions to IPv6.

# 24

## **IPV6 OVERVIEW, CHANGES, AND TRANSITION**



Internet Protocol version 6 (IPv6) is destined to be the future of IP, and due to IP's critical importance, it will form the basis for the future of TCP/IP and the Internet as well. In fact, it's been under development since the middle of the last decade, and a real IPv6 internetwork has been used for testing for a number of years as well. Despite this, many people don't know much about IPv6, other than the fact that it's a newer version of IP. Some have never even heard of it at all! I'm going to rectify that, of course—but before I delve into the important changes made in IPv6 addressing, packaging, fragmentation, and other functions, let's start with a bird's-eye view of IPv6.

In this chapter, I provide a brief higher-level overview of IPv6, including a look at how it differs from IP version 4 (IPv4) in general terms. I begin with a brief overview of IPv6 and why it was created. I list the major changes made in IPv6 and the new additions to the protocol. I also explain some of the difficulties associated with transitioning the enormous global Internet from IPv4 to IPv6.

## IPv6 Motivation and Overview

“If it ain’t broke, don’t fix it.” This is one of my favorite pieces of folk wisdom. I generally like to stick with what works, as do most people. And IPv4 works pretty darned well. It’s been around for decades now and has survived the growth of the Internet from a small research network into a globe-spanning powerhouse. So, like a trusty older car that you’ve operated successfully for years, why should you replace it if it still gets the job done?

Like that older car, you could continue to use IPv4 for the foreseeable future. The question is: at what cost? An older car can be kept in good working order if you are willing to devote the time and money it takes to maintain and service it. However, it will still be limited in some of its capabilities. Its reliability may be suspect. It won’t have the latest features. With the exception of those who like to work on cars as a hobby, it eventually stops making sense to keep fixing up an older vehicle.

In some ways, this isn’t that great of an analogy. Our highways aren’t all that much different than they were in the 1970s, and most other issues related to driving a car haven’t changed all that much in the past 25 years either. The choice of updating a vehicle or not is based on practical considerations more than necessity.

In contrast, look at what has happened to the computer and networking worlds in the last 25 years! Today’s handheld PCs can do more than the most powerful servers could back then. Networking technologies are 100 or even 1,000 times as fast. The number of people connecting to the global Internet has increased by an even larger factor. And the ways that computers communicate have, in many cases, changed dramatically.

IPv4 could be considered in some ways like an older car that has been meticulously maintained and repaired over time. It gets the job done, but its age is starting to show. The main problem with IPv4 is its relatively small address space, a legacy of the decision to use only 32 bits for the IP address. Under the original classful addressing allocation scheme, we would have probably already run out of IPv4 addresses by now. Moving to classless addressing has helped postpone this, as have technologies like IP Network Address Translation (NAT), which allows privately addressed hosts to access the Internet.

In the end, however, these represent patch jobs and imperfect repairs applied to keep the aging IPv4 automobile on the road. The core problem, the 32-bit address space that is too small for the current and future size of the Internet, can be solved only by moving to a larger address space. This was the primary motivating factor in creating the next version of IP, *IPv6*.

**NOTE** *The reason why the successor to IPv4 is version 6 and not version 5 is because version number 5 was used to refer to an experimental protocol called the Internet Stream Protocol, which was never widely deployed. See Chapter 15 for a full discussion of IP history and versions.*

## IPv6 Standards

IPv6 represents the first major change to IP since IPv4 was formalized in 1981. For many years, its core operation was defined in a series of RFCs published in 1998: RFCs 2460 through 2467. The most notable of these are the main IPv6 standard, RFC 2460, “Internet Protocol, Version 6 (IPv6) Specification,” and documents

describing the two helper protocols for IPv6: RFC 2461, which describes the IPv6 Neighbor Discovery Protocol (ND), and RFC 2463, which describes Internet Control Message Protocol version 6 (ICMPv6) for IPv6.

In addition to these, two documents were also written in 1998. They discuss more about IP addressing: RFC 2373, “IP Version 6 Addressing Architecture,” and RFC 2374, “An IPv6 Aggregatable Global Unicast Address Format.” Due to changes in how IPv6 addressing was to be implemented, these were updated in 2003 by RFC 3513, “Internet Protocol Version 6 (IPv6) Addressing Architecture,” and RFC 3587, “IPv6 Global Unicast Address Format.”

Many other RFCs define more specifics of how IPv6 works, and many also describe IPv6-compatible versions of other TCP/IP protocols like the Domain Name System (DNS; see Chapter 52) and Dynamic Host Control Protocol (DHCP; see Chapter 61). IPv6 is still very much a work in progress, with new standards being proposed and adopted on a regular basis.

Because IPv6 is the version of IP that’s designed for the next generation of the Internet, it is also sometimes called *IP Next Generation* or *IPng*. Personally, I don’t care for this name; it reminds me too much of *Star Trek: The Next Generation*.

Regardless of its name, IPv6 or IPng was designed to take TCP/IP and the Internet “where none have gone before.” (Sorry, I *had* to!)

## **Design Goals of IPv6**

The problem of addressing was the main motivation for creating IPv6. Unfortunately, this has caused many people to think that the address space expansion is the *only* change made in IP, which is definitely not the case. Since making a change to IP is such a big deal, it’s something done rarely. It made sense to correct not just the addressing issue, but also to update the protocol in a number of other respects in order to ensure its viability. In fact, even the addressing changes in IPv6 go far beyond just adding more bits to IP address fields.

Some of the most important goals in designing IPv6 include the following:

**Larger Address Space** IPv6 needed to provide more addresses for the growing Internet.

**Better Management of Address Space** Developers wanted IPv6 to include not only more addresses, but also a more capable way of dividing the address space and using the bits in each address.

**Elimination of Addressing Kludges** Technologies like NAT are effectively kludges that make up for the lack of address space in IPv4. IPv6 eliminates the need for NAT and similar work-arounds, allowing every TCP/IP device to have a public address.

**Easier TCP/IP Administration** The designers of IPv6 hoped to resolve some of the current labor-intensive requirements of IPv4, such as the need to configure IP addresses. Even though tools like DHCP eliminate the need to manually configure many hosts, it only partially solves the problem.

**Modern Design for Routing** In contrast to IPv4, which was designed before anyone had an idea what the modern Internet would be like, IPv6 was created specifically for efficient routing in the current Internet, and with the flexibility for the future.

**Better Support for Multicasting** Multicasting was an option in IPv4 from the start, but support for it has been slow in coming.

**Better Support for Security** IPv4 was designed at a time when security wasn't much of an issue because there were a relatively small number of networks on the Internet, and those networks' administrators often knew each other. Today, security on the public Internet is a big issue, and the future success of the Internet requires that security concerns be resolved.

**Better Support for Mobility** When IPv4 was created, there really was no concept of mobile IP devices. The problems associated with computers that move between networks led to the need for Mobile IP. IPv6 builds on Mobile IP and provides mobility support within IP itself.

**KEY CONCEPT** The new version of the IP is *Internet Protocol version 6 (IPv6)*. It was created to correct some of the significant problems of IPv4, especially the looming deficiency of the IPv4 address space, to improve the operation of the protocol as a whole, and to take TCP/IP into the future.

At the same time that IPv6 was intended to address these and many other issues with traditional IP, its changes are nevertheless *evolutionary*, not *revolutionary*. During the many discussions in the Internet Engineering Task Force (IETF) in the 1990s, there were some who said that while we were updating IP, perhaps we should make a complete, radical change to a new type of internetworking protocol completely. The end decision was not to do this, but to define a more capable version of the IP that we've been using all along.

The reason for this is simple: IP, like our trusted older car, *works*. IPv6 represents an update that strives to add to the best characteristics of IPv4, rather than making everyone start over from scratch with something new and unproven. This design ensures that whatever pain may result from the change from IPv4 to IPv6 can be managed, and hopefully, minimized.

## Major Changes and Additions in IPv6

In the preceding overview, I explained that the primary motivation for creating a new version of IP was to fix the problems with addressing under IPv4. But as you also saw, numerous other design goals existed for the new protocol as well. Once the decision was made to take the significant step of creating a new version of a protocol as important as IP, it made sense to use the opportunity to make as many improvements as possible.

Of course, there is still the problem of the pain of change to worry about, so each potential change or addition in IPv6 needed to have benefits that would outweigh its costs. The resulting design does a good job of providing useful

advantages while maintaining most of the core of the original IP. The following are some of the most important changes between IPv4 and IPv6, and they demonstrate some of the ways that the IPv6 team met the design goals for the new protocol:

**Larger Address Space** IPv6 addresses are 128 bits long instead of 32 bits. This expands the address space from around 4 billion addresses to, well, an astronomical number (over 300 trillion trillion addresses).

**Hierarchical Address Space** One reason why the IPv6 address size was expanded so much was to allow it to be hierarchically divided to provide a large number of many classes of addresses.

**Hierarchical Assignment of Unicast Addresses** A special global unicast address format was created to allow addresses to be easily allocated across the entire Internet. It allows for multiple levels of network and subnetwork hierarchies at both the Internet service provider (ISP) and the organizational level. It also permits the generation of IP addresses based on underlying hardware interface device IDs such as Ethernet MAC addresses.

**Better Support for Nonunicast Addressing** Support for multicasting is improved, and support for a new type of addressing, *anycast* addressing, has been added. This new kind of addressing basically says, “Deliver this message to the easiest-to-reach member of this group,” and potentially enables new types of messaging functionality.

**Autoconfiguration and Renumbering** A provision is included to allow easier auto-configuration of hosts and renumbering of the IP addresses in networks and sub-networks as needed. A technique also exists for renumbering router addresses.

**New Datagram Format** The IP datagram format has been redefined and given new capabilities. The main header of each IP datagram has been streamlined, and support has been added for the ability to easily extend the header for datagrams that require more control information.

**Support for Quality of Service (QoS)** IPv6 datagrams include QoS features that allow for better support for multimedia and other applications that require QoS.

**Security Support** Security support is designed into IPv6 using the authentication and encryption extension headers and other features.

**Updated Fragmentation and Reassembly Procedures** The way that the fragmentation and reassembly of datagrams works has been changed in IPv6. The improved routing efficiency better reflects the realities of today’s networks.

**Modernized Routing Support** IPv6 is designed to support modern routing systems and allow for expansion as the Internet grows.

**Transition Capabilities** Since it was recognized from the start that going from IPv4 to IPv6 is a big move, support for the IPv4/IPv6 transition has been provided in numerous areas. This includes a plan for interoperating IPv4 and IPv6 networks, for mapping between IPv4 and IPv6 addresses, and other transition support.

**Changes to Other Protocols** With the introduction of IPv6, several other TCP/IP protocols that deal intimately with IP have also had to be updated. One of these is ICMP, the most important support protocol for IPv4, which has been revised through the creation of ICMPv6 for IPv6. An addition to TCP/IP is the ND protocol, which performs several functions for IPv6 that were done by the Address Resolution Protocol (ARP) and ICMP in version 4.

The following chapters on IPv6 provide much more detail on these changes and additions to IP. You'll notice that the majority of these are related to addressing, because that is where the greatest number of important changes were made in IPv6. Of course, routing and addressing are closely related, and the changes to addressing have had a big impact on routing as well.

## Transition from IPv4 to IPv6

IP is the foundation of the TCP/IP protocol suite and the Internet, and thus it's somewhat comparable to the foundation of a house in terms of its structural importance. Given this, changing IP is somewhat analogous to making a substantial modification to the foundation of your house. Since IP is used to connect together many devices, it is like changing not just your house, but every house in the world!

How do you change the foundation of a house? Very carefully. The same caution is required with the implementation of IPv6. While most people think IPv6 is something new, the reality is that the planning and development of IPv6 has been underway for nearly a full decade, and if we were starting from scratch, the protocol would have been ready for action years ago. However, there is a truly enormous installed base of IPv4 hardware and software. This means the folks who develop TCP/IP could not just flip a switch and have everyone move over to using IPv6. Instead, a *transition* from IPv4 to IPv6 had to be planned.

The transition is already under way, though most people don't know about it. As I said, development of IPv6 itself is pretty much complete, though work continues on refining the protocol and also on the development of IPv6-compatible versions of other protocols. The implementation of IPv6 began with the creation of development networks to test IPv6's operation. These were connected together to form an experimental IPv6 internetwork called the *6BONE* (which is a contraction of the phrase *IPv6 backbone*). This internetwork has been in operation for several years.

### ***IPv4 to IPv6 Transition: Differences of Opinion***

Experimental networks are well and good, but the big issue is transitioning the Internet to IPv6, and here, opinion diverges rather quickly. In one camp are the corporations, organizations, and individuals. All of these groups are quite eager to transition to IPv6 quickly in order to gain the many benefits it promises in the areas of addressing, routing, and security. Others are taking a much more cautious approach, noting that the dire predictions in the mid-1990s of IPv4's imminent doom have not come to pass, and arguing that we should take our time to make sure IPv6 is going to work on a large scale.

These two groups will continue to play tug-of-war for the next few years, but it seems that the tide is now turning toward those who want to speed up the now-years-long transition. The move toward adoption of IPv6 as a *production* protocol is being spearheaded by a number of groups and organizations. IPv6 has a lot of support in areas outside the United States, many of which are running short of IPv4 addresses due to small allocations relative to their size. One such area is Asia, a region with billions of people, rapidly growing Internet use, and a shortage of IPv4 addresses.

Within the United States, which has the lion's share of IPv4 addresses (because the Internet was developed here), there seems to be a bit less enthusiasm for rapid IPv6 deployment. Even here, however, IPv6 got a major shot in the arm in July 2003 when the United States Department of Defense (DoD) announced that starting in October of that year, it would purchase only networking products that included compatibility with IPv6. The DoD (which was responsible for the development of the Internet in the first place) hopes to be fully transitioned to IPv6 by 2008. This will likely have a big impact on the plans of other governmental and private organizations in the United States.

The creators of IPv6 knew from the start that transition was going to be an important issue with the new protocol. IPv6 is not compatible with IPv4 because the addressing system and datagram format are different. Yet the IPv6 designers knew that since the transition would take many years, it was necessary that they provide a way for IPv4 and IPv6 hosts to interoperate. Consider that in any transition there are always stragglers. Like the old Windows 3.11 PC in the corner that you still need to use once in a while, some devices will remain on IPv4, even when most of the Internet is IPv6, because they were never upgraded.

**KEY CONCEPT** Due to the many differences between IPv4 and IPv6, and the fundamental importance of IP to TCP/IP, an orderly transition has been planned from IPv4 to IPv6 over a period of many years.

## **IPv4 to IPv6 Transition Methods**

The IETF has been working on specific provisions to allow a smooth transition from IPv4 to IPv6, and hardware and software interoperability solutions to let newer IPv6 devices access IPv4 hosts. A technique was included in IPv6 to allow administrators to embed IPv4 addresses within IPv6 addresses. Special methods are defined to handle interoperability, including the following:

**Dual-Stack Devices** Routers and some other devices may be programmed with both IPv4 and IPv6 implementations to allow them to communicate with both types of hosts.

**IPv4/IPv6 Translation** Dual-stack devices may be designed to accept requests from IPv6 hosts, convert them to IPv4 datagrams, send the datagrams to the IPv4 destination, and then process the return datagrams similarly.

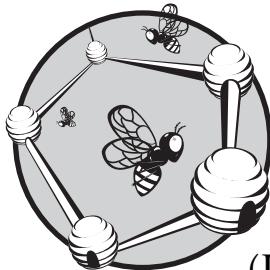
**IPv4 Tunneling of IPv6** IPv6 devices that don't have a path between them consisting entirely of IPv6-capable routers may be able to communicate by encapsulating IPv6 datagrams within IPv4. In essence, they would be using IPv6 on top of IPv4; that is, two network layers. The encapsulated IPv4 datagrams would travel across conventional IPv4 routers.

Bear in mind that these solutions generally address only backward compatibility to allow IPv6 devices to talk to IPv4 hardware. Forward compatibility between IPv4 and IPv6 is not possible because IPv4 hosts cannot communicate with IPv6 hosts; they lack the knowledge of how IPv6 works. It is possible that certain special adaptations might be created to allow IPv4 hosts to access IPv6 hosts. But eventually, all IPv4 devices of any importance will want to migrate to IPv6.

The IETF has done such a good job in the past with introducing new technologies, and so much effort has been put into the IPv6 transition, that I am quite confident that the transition to IPv6 will come off with few, if any, problems. One good thing about the transition is that IPv4 is, at the present time, still getting the job done, so there is no big hurry to make the move to IPv6. While technologies such as CIDR and NAT are like Band-Aids on IPv4, they have been very successful ones in extending the useful life of the aging protocol.

# 25

## IPV6 ADDRESSING



The primary motivation for creating Internet Protocol version 6 (IPv6) was to rectify the addressing problems in version 4 (IPv4). Along with acquiring more addresses, the IPv6 designers desired a way of interpreting, assigning, and using addresses in a way that was more consonant with modern internetworking. So, it's no surprise that many of the changes in IPv6 are associated with IP addressing. The IPv6 addressing scheme is similar in concept to IPv4 addressing, but has been completely overhauled to create an addressing system that's capable of supporting continued Internet expansion and new applications for the foreseeable future.

This chapter describes the concepts and methods associated with addressing under IPv6. I begin with a look at some addressing generalities in IPv6, including the addressing model, address types' size, and address space. I discuss the unique and sometimes confusing representations and notations used for IPv6 addresses and prefixes. Then I look at how addresses are arranged and allocated into types, beginning with an overall look at address space composition and then at the global unicast address format. I describe the new methods used for mapping IP addresses to underlying physical

network addresses. I then describe special IPv6 addressing issues, including reserved and private addresses, IPv4 address embedding, anycast and multicast addresses, and autoconfiguration and renumbering of addresses.

Addressing under IPv6 is outlined in the main IPv6 RFC, RFC 2460, “Internet Protocol, Version 6 (IPv6) Specification.” However, most of the details of IPv6 addressing are contained in two other standards: RFC 3513, “Internet Protocol Version 6 (IPv6) Addressing Architecture,” and RFC 3587, “IPv6 Global Unicast Address Format.” These replaced the 1998 standards RFC 2373, “IP Version 6 Addressing Architecture,” and RFC 2374, “An IPv6 Aggregatable Global Unicast Address Format.”

**BACKGROUND INFORMATION** As with the other IPv6 chapters in this book, my look at addressing is based somewhat on a contrast to how addressing is done in IPv4. I strongly recommend a thorough understanding of IPv4 addressing, including classless addressing using Classless Inter-Domain Routing (CIDR), as presented in Chapters 16 through 23, before proceeding here. As with the IPv4 addressing sections, familiarity with how binary numbers work, and conversion between binary and decimal numbers is also a good idea. Chapter 4, which provides some background on data representation and the mathematics of computing, may be of assistance in that respect.

## IPv6 Addressing Overview: Addressing Model, Address Types, and Address Size

As you saw in the previous chapter, IPv6 represents a significant update to IP, but its modifications and additions are made without changing the core nature of how IP works. Addressing is the place where most of the differences between IPv4 and IPv6 are seen, but the changes are mostly in how addresses are implemented and used. The overall model used for IP addressing in IPv6 is pretty much the same as it was in IPv4; some aspects have not changed at all, while others have changed only slightly.

### IPv6 Addressing Model Characteristics

Here are some of the general characteristics of the IPv6 addressing model that are basically the same as in IPv4:

**Core Functions of Addressing** The two main functions of addressing are still network interface identification and routing. Routing is facilitated through the structure of addresses on the internetwork.

**Network Layer Addressing** IPv6 addresses are still the ones associated with the network layer in TCP/IP networks and are distinct from data link layer (also sometimes called *physical*) addresses.

**Number of IP Addresses per Device** Addresses are still assigned to network interfaces, so a regular host like a PC will usually have one (unicast) address, and routers will have more than one for each of the physical networks to which it connects.

**Address Interpretation and Prefix Representation** IPv6 addresses are like classless IPv4 addresses in that they are interpreted as having a network identifier part and a host identifier part (a network ID and a host ID), but that the delineation is not encoded into the address itself. A prefix-length number, using CIDR-like notation, is used to indicate the length of the network ID (prefix length).

**Private and Public Addresses** Both types of addresses exist in IPv6, though they are defined and used somewhat differently.

## **IPv6 Supported Address Types**

One important change in the addressing model of IPv6 is the *address types* supported. IPv4 supported three address types: unicast, multicast, and broadcast. Of these, the vast majority of actual traffic was unicast. IP multicast support was not widely deployed until many years after the Internet was established and it continues to be hampered by various issues. Use of broadcast in IP had to be severely restricted for performance reasons (we don't want any device to be able to broadcast across the entire Internet!).

IPv6 also supports three address types, but with the following changes:

**Unicast Addresses** These are standard unicast addresses as in IPv4, one per host interface.

**Multicast Addresses** These are addresses that represent various groups of IP devices. A message sent to a multicast address goes to all devices in the group. IPv6 includes much better multicast features and many more multicast addresses than IPv4. Since multicast under IPv4 was hampered in large part due to lack of support of the feature by many hardware devices, support for multicasting is a required, not optional, part of IPv6.

**Anycast Addresses** Anycast addressing is used when a message must be sent to any member of a group, but does not need to be sent to all of them. Usually the member of the group that is easiest to reach will be sent the message. One common example of how anycast addressing could be used is in load sharing among a group of routers in an organization.

Broadcast addressing as a distinct addressing method is gone in IPv6. Broadcast functionality is implemented using multicast addressing to groups of devices. A multicast group to which all nodes belong can be used for broadcasting in a network, for example.

**KEY CONCEPT** IPv6 has unicast and multicast addresses like IPv4. There is, however, no distinct concept of a broadcast address in IPv6. A new type of address, the *anycast* address, has been added to allow a message to be sent to any one member of a group of devices.

An important implication of the creation of anycast addressing is removal of the strict uniqueness requirement for IP addresses. Anycast is accomplished by assigning the same IP address to more than one device. The devices must also be specifically told that they are sharing an anycast address, but the addresses themselves are structurally the same as unicast addresses.

The bulk of the remainder of this chapter focuses on unicast addressing, since it is by far the most important type. Multicast and anycast addressing are given special attention in a separate section later in this chapter.

## ***IPv6 Address Size and Address Space***

Of all the changes introduced in IPv6, easily the most celebrated is the increase in the size of IP addresses, which resulted in a corresponding massive increase in the size of the address space as well. It's not surprising that these sizes were increased compared to IPv4—everyone has known for years that the IPv4 address space was too small to support the future of the Internet. What's remarkable is the level of increase and the implications for how Internet addresses are used.

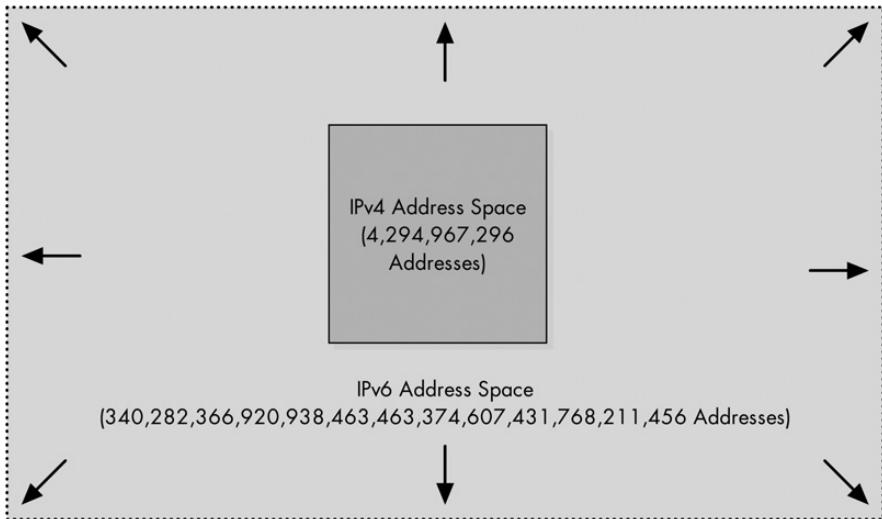
In IPv4, IP addresses are 32 bits long; these are usually grouped into 4 octets of 8 bits each. The theoretical IPv4 address space is  $2^{32}$ , or 4,294,967,296 addresses. To increase this address space, we simply increase the size of addresses; each extra bit we give to the address size doubles the address space. Based on this, some folks expected the IPv6 address size to increase from 32 to 48 bits, or perhaps 64 bits. Either of these numbers would have given a rather large number of addresses.

However, IPv6 addressing doesn't use either of these figures. Instead, the IP address size jumps all the way to 128 bits, or 16 8-bit octets/bytes. The size of the IPv6 address space is, quite literally, astronomical. Like the numbers that describe the number of stars in a galaxy or the distance to the furthest pulsars, the number of addresses that can be supported in IPv6 is mind-boggling. See Figure 25-1 for an idea of what I mean by *astronomical*.

Since IPv6 addresses are 128 bits long, the theoretical address space, if all addresses were used, is  $2^{128}$  addresses. This number, when expanded out, is 340,282,366,920,938,463,463,374,607,431,768,211,456, which is normally expressed in scientific notation as about  $3.4 \times 10^{38}$  addresses. Whoa! That's about 340 trillion, *trillion, trillion* addresses. As I said, it's pretty hard to grasp just how large this number is. Consider these comparisons:

- It's enough addresses for many trillions of addresses to be assigned to every human being on the planet.
- The Earth is about 4.5 billion years old. If you had been assigning IPv6 addresses at a rate of 1 billion per second since the Earth was formed, you would have by now used up less than one trillionth of the address space.
- The Earth's surface area is about 510 trillion square meters. If a typical computer has a footprint of about one-tenth of a square meter, you would have to stack computers 10 billion high—blanketing the entire surface of the Earth—to use up that same trillionth of the address space.

OK, I think you get the idea. It's clear that one goal of the decision to go to 128-bit addresses is to make sure that we will never run out of address space again, and it seems quite likely that this will be the case.



**Figure 25-1: A (poor) representation of relative IPv4 and IPv6 address space sizes** I wanted to make a cool graphic to show the relative sizes of the IPv4 and IPv6 address spaces. You know, where I would show the IPv6 address space as a big box and the IPv4 address space as a tiny one. The problem is that the IPv6 address space is so much larger than the IPv4 space that there is no way to show it to scale! To make this diagram to scale, imagine the IPv4 address space is the 1.6-inch square above. In that case, the IPv6 address space would be represented by a square the size of the solar system!

There are drawbacks to having such a huge address space, too. Consider that even with a 64-bit address, we would have a very large address space;  $2^{64}$  equals 18,446,744,073,709,551,616, or about 18 million trillion. These are still probably more addresses than the Internet will ever need. However, by going to 128 bits instead, this has made dealing with IP addresses unruly (as you'll see in the next section). This has also increased overhead, since every datagram header or other place where IP addresses are referenced must use 16 bytes for each address instead of the 4 bytes that were needed in IPv4, or the 8 bytes that might have been required with a 64-bit address.

**KEY CONCEPT** The IPv6 address space is really, *really* big!

So why the overkill of going to 128 bits? The main reason is *flexibility*. Even though you can have a couple zillion addresses if we allocate them one at a time, this makes assignment difficult. The developers got rid of class-oriented addressing in IPv4 because it wasted address space. The reality, though, is that being able to waste address space is a useful luxury.

Having 128 bits allows us to divide the address space and assign various purposes to different bit ranges, while still not having to worry about running out of space. Later in this chapter, in the section describing the IPv6 global unicast address format, you'll see one way that those 128 bits are put to good use: They allow you to create a hierarchy of networks while still saving 64 bits for host IDs. This hierarchy has its own advantages.

# **IPv6 Address and Address Notation and Prefix Representation**

Increasing the size of IP addresses from 32 bits to 128 bits expands the address space to a gargantuan size, thereby ensuring that we will never again run out of IP addresses, and thereby allowing flexibility in how they are assigned and used. Unfortunately, there are some drawbacks to this method, and one of them is that 128-bit numbers are very large. The size makes them awkward and difficult to use.

Computers work in binary, and they have no problem dealing with long strings of ones and zeros, but humans find them confusing. Even the 32-bit addresses of IPv4 are cumbersome for us to deal with, which is why we use dotted decimal notation for them unless we need to work in binary (as with subnetting). However, IPv6 addresses are so much larger than IPv4 addresses that it becomes problematic to use dotted decimal notation. To use this notation, we would split the 128 bits into 16 octets and represent each with a decimal number from 0 to 255. However, we would end up not with 4 of these numbers, but 16. A typical IPv6 address in this notation would appear as follows:

128.91.45.157.220.40.0.0.0.0.252.87.212.200.31.255

The binary and dotted decimal representations of this address are shown near the top of Figure 25-2. In either case, the word *elegant* doesn't exactly spring to mind.

**Figure 25-2: Binary, decimal, and hexadecimal representations of IPv6 addresses** The top two rows show binary and dotted decimal representations of an IPv6 address; neither is commonly used (other than by computers themselves!). The top row of the lower table shows the full hexadecimal representation, while the next two rows illustrate zero suppression and compression. The last row shows mixed notation, with the final 32 bits of an IPv6 address shown in dotted decimal notation (212.200.31.255). This is most commonly used for embedded IPv4 addresses.

## ***IPv6 Address Hexadecimal Notation***

To make addresses shorter, the decision was made in IPv6 to change the primary method of expressing addresses to use hexadecimal instead of decimal. The advantage of this is that it requires fewer characters to represent an address, and

converting from hexadecimal to binary and back again is much easier than converting from binary to decimal or vice versa. The disadvantage is that many people find hexadecimal difficult to comprehend and work with, especially because the notion of 16 values in each digit is a bit strange.

The hexadecimal notation used for IPv6 addresses is similar to the same method used for IEEE 802 MAC addresses, and for technologies like Ethernet. With these MAC addresses, 48 bits are represented by 6 octets, each octet being a hexadecimal number from 0 to FF, separated by a dash or colon, like this:

0A-A7-94-07-CB-D0

Since IPv6 addresses are larger, they are instead grouped into eight 16-bit *words*, separated by colons, to create what is sometimes called *colon hexadecimal notation*, as shown in Figure 25-2. So, the IPv6 address given in the previous example would be expressed as follows:

805B:2D9D:DC28:0000:0000:FC57:D4C8:1FFF

To keep the address size down, leading zeros can be suppressed in the notation so you can immediately reduce this to the following:

805B:2D9D:DC28:0:0:FC57:D4C8:1FFF

Well, it's definitely shorter than dotted decimal, but still pretty long. When you are dealing with numbers this big, there's only so much you can do. This is part of why the use of Domain Name System (DNS) names for hosts becomes much more important under IPv6 than it is in IPv4: Who could remember a hex address that long?

### **Zero Compression in IPv6 Addresses**

Fortunately, there is a shortcut that can be applied to shorten some addresses even further. This technique is sometimes called *zero compression*. The method allows a single string of contiguous zeros in an IPv6 address to be replaced by double colons. So, for example, the previous address could be expressed as follows:

805B:2D9D:DC28::FC57:D4C8:1FFF

You know how many zeros are replaced by the two colons (::) because you can see how many fully expressed (uncompressed) hexadecimal words are in the address. In this case, there are six, so the :: represents two zero words. To prevent ambiguity, the double colons can appear only once in any IP address, because if it appeared more than once, you could not tell how many zeros were replaced in each instance. So, if the example address were 805B:2D9D:DC28:0:0:FC57:0:0, you could replace either the first pair of zeros or the second, but not both.

Zero compression doesn't make the example much shorter, but due to how IPv6 addresses are structured, long strings of zeros are common. For example, consider this address:

FF00:4501:0:0:0:0:0:32

With compression, this could be shortened as follows:

FF00:4501::32

The technique works even better on special addresses. The full IPv6 loopback address is written as follows:

0:0:0:0:0:0:1

With compression, the loopback address looks like this:

::1

For even more fun, consider the especially odd IPv6 unspecified address, as shown here:

0:0:0:0:0:0:0

Apply zero compression to an address that is all zeros, and what do you get?

::

No numbers at all! Of course, thinking of :: as an address *does* take some getting used to.

**KEY CONCEPT** For brevity, IPv6 addresses are represented using eight sets of four hexadecimal digits, a form called *colon hexadecimal notation*. Additional techniques, called *zero suppression* and *zero compression*, are used to reduce the size of displayed addresses further by removing unnecessary zeros from the presentation of the address.

## IPv6 Mixed Notation

There is also an alternative notation used in some cases, especially for expressing IPv6 addresses that embed IPv4 addresses (discussed later in this chapter). For these, it is useful to show the IPv4 portion of the address in the older dotted decimal notation, since that's what you use for IPv4. Since embedding uses the last 32 bits for the IPv4 address, the notation has the first 96 bits in colon hexadecimal notation and the last 32 bits in dotted decimal. So, to take the earlier example again, in *mixed notation* it would be shown as follows:

805B:2D9D:DC28::FC57:**212.200.31.255**

This isn't really a great example of mixed notation, because embedding usually involves long strings of zeros followed by the IPv4 address. Thus, zero compression comes in very handy here. Instead of seeing something like this:

0:0:0:0:0:212.200.31.255

You will typically see this:

::212.200.31.255

At first glance, this appears to be an IPv4 address. You must keep a close eye on those colons in IPv6!

**KEY CONCEPT** A special mixed notation is defined for IPv6 addresses whose last 32 bits contain an embedded IPv4 address. In this notation, the first 96 bits are displayed in regular colon hexadecimal notation, and the last 32 bits are displayed in IPv4-style dotted decimal.

## IPv6 Address Prefix Length Representation

Like IPv4 classless addresses, IPv6 addresses are fundamentally divided into a number of network ID bits followed by a number of host ID bits. The network identifier is called the *prefix*, and the number of bits used is the *prefix length*. This prefix is represented by adding a slash after the address and then putting the prefix length after the slash. This is the same method used for classless IPv4 addressing with CIDR. For example, if the first 48 bits of the sample address were the network ID (prefix), then we would express this as 805B:2D9D:DC28::FC57:D4C8:1FFF/48.

**KEY CONCEPT** In IPv6, the size of an address's prefix is indicated by the prefix length that follows the address, separated with a slash, just as it is done in IPv4 classless addressing.

As in IPv4, specifiers for whole networks will typically end in long strings of zeros. These can be replaced by double colons (::) using zero compression. For example, the 48-bit network ID for the previous example is 805B:2D9D:DC28:0:0:0:0/48, or 805B:2D9D:DC28::/48. You *must* include the “::” if replacing the trailing zeros.

## IPv6 Address Space Allocation

After dealing for so many years with the very small IPv4 address space, the enormous number of addresses in IPv6 must have made the Internet Engineering Task Force (IETF) engineers feel like kids in a candy shop. They were good kids, however, and didn't run wild, grabbing all the candy they could find and gobbling it up. They very carefully considered how to divide the address space for various uses. Of course, when you have this much candy, sharing becomes pretty easy.

As was the case with IPv4, the two primary concerns in deciding how to divide the IPv6 address space were address assignment and routing. The designers of IPv6 wanted to structure the address space to make allocation of addresses to Internet service providers (ISPs), organizations, and individuals as easy as possible.

At first, perhaps ironically, this led the creators of IPv6 back full circle to the use of specific bit sequences to identify different types of addresses, just like the old classful addressing scheme. The address type was indicated by a set of bits at the start of the address, called the format prefix (FP). The format prefix was conceptually identical to the one to four bits used in IPv4 classful addressing to denote address classes, but was variable in length, ranging from three to ten bits. Format prefixes were described in RFC 2373.

In the years following the publication of RFC 2373, the gurus who run the Internet had a change of heart regarding how address blocks should be considered. They still wanted to divide the IPv6 address space into variably sized blocks for different purposes. However, they realized that many people were starting to consider the use of format prefixes to be equivalent to the old class-oriented IPv4

system. Their main concern was that implementers might program into IPv6 hardware logic to make routing decisions based only on the first few bits of the address. This was specifically *not* how IPv6 is supposed to work; for one thing, the allocations are subject to change.

Thus, one of the modifications made in RFC 3513 was to change the language regarding IPv6 address allocations, and specifically, to remove the term *format prefix* from the standard. The allocation of different parts of the address space is still done based on particular patterns of the first three to ten bits of the address to allow certain categories to have more addresses than others. The elimination of the specific term denoting this is intended to convey that these bits should not be given special attention.

Table 25-1 shows the allocations of the IPv6 address space and what fraction of the total address space each represents.

**Table 25-1:** IPv6 Address Space Allocations

Leading Bits	Fraction of Total IPv6 Address Space	Allocation
0000 0000	1/256	Unassigned (Includes special addresses such as the unspecified and loopback addresses)
0000 0001	1/256	Unassigned
0000 001	1/128	Reserved for NSAP address allocation
0000 01	1/64	Unassigned
0000 1	1/32	Unassigned
0001	1/16	Unassigned
001	1/8	Global unicast addresses
010	1/8	Unassigned
011	1/8	Unassigned
100	1/8	Unassigned
101	1/8	Unassigned
110	1/8	Unassigned
1110	1/16	Unassigned
1111 0	1/32	Unassigned
1111 10	1/64	Unassigned
1111 110	1/128	Unassigned
1111 1110 0	1/512	Unassigned
1111 1110 10	1/1024	Link-local unicast addresses
1111 1110 11	1/1024	Site-local unicast addresses
1111 1111	1/256	Multicast addresses

This is more complicated than the IPv4 classful scheme because there are so many more categories and they range greatly in size, even if most of them are currently unassigned.

An easier way to make sense of this table is to consider the division of the IPv6 address space into *eighths*. Of these eight groups, one (001) has been reserved for unicast addresses; a second (000) has been used to carve out smaller reserved blocks, and a third (111) has been used for sub-blocks for local and multicast addresses. Five are completely unassigned.

You can see that the IPv6 designers have taken great care to allocate only the portion of these “eighths” of the address space that they felt was needed for each type of address. For example, only a small portion of the part of the address space beginning 111 was used, with most of it left aside. In total, only 71/512ths of the address space is assigned right now, or about 14 percent. The other 86 percent is unassigned and kept aside for future use. (Bear in mind that even 1/1024th of the IPv6 address space is gargantuan—it represents trillions of trillions of addresses.)

Later sections in this chapter provide more information on several of these address blocks. Note that the 0000 0000 reserved block is used for several special address types, including the loopback address, the unspecified address, and IPv4 address embedding. The 1111 1111 format prefix identifies multicast addresses; this string is FF in hexadecimal, so any address beginning with FF is a multicast address in IPv6.

## IPv6 Global Unicast Address Format

It is anticipated that unicast addressing will be used for the vast majority of Internet traffic under IPv6, as is the case for IPv4. It is for this reason that the largest of the assigned blocks of the IPv6 address space is dedicated to unicast addressing. A full one-eighth slice of the enormous IPv6 address “pie” is assigned to unicast addresses, which are indicated by a 001 in the first three bits of the address. The question is: How do we use the remaining 125 bits in the spacious IP addresses?

### Rationale for a Structured Unicast Address Block

When IPv4 was first created, the Internet was rather small, and the model for allocating address blocks was based on a central coordinator: the Internet Assigned Numbers Authority (IANA). Everyone who wanted address blocks would go straight to the central authority. As the Internet grew, this model became impractical. Today, IPv4’s classless addressing scheme allows variable-length network IDs and hierarchical assignment of address blocks. Big ISPs get large blocks from the central authority, and then subdivide them and allocate them to their customers, and so on. This is managed by today’s ISPs, but there is nothing in the address space that helps manage the allocation process. In turn, each organization has the ability to further subdivide its address allocation to suit its internal requirements.

The designers of IPv6 had the benefit of this experience and realized there would be tremendous advantages to designing the unicast address structure to reflect the overall topology of the Internet. These include the following:

- Easier allocation of address blocks at various levels of the Internet topological hierarchy.
- IP network addresses that automatically reflect the hierarchy by which routers move information across the Internet, thereby allowing routes to be easily aggregated for more efficient routing.

- Flexibility for organizations like ISPs to subdivide their address blocks for customers.
- Flexibility for end-user organizations to subdivide their address blocks to match internal networks, much as subnetting did in IPv4.
- Greater meaning to IP addresses. Instead of just being a string of 128 bits with no structure, it would become possible to look at an address and know certain things about it.

### ***Generic Division of the Unicast Address Space***

The most generic way of dividing up the 128 bits of the unicast address space is into three sections, as shown in Table 25-2.

**Table 25-2:** Generic IPv6 Global Unicast Address Format

Field Name	Size (Bits)	Description
Prefix	$n$	Global Routing Prefix: The network ID or prefix of the address, used for routing.
Subnet ID	$m$	Subnet Identifier: A number that identifies a subnet within the site.
Interface ID	$128-n-m$	Interface Identifier: The unique identifier for a particular interface (host or other device). It is unique within the specific prefix and subnet.

The *global routing prefix* and *subnet identifier* represent the two basic levels at which addresses need to be hierarchically constructed: that is, global and site-specific. The routing prefix consists of a number of bits that can be further subdivided according to the needs of Internet registries and ISPs. This subdivision reflects the topography of the Internet as a whole. The subnet ID gives a number of bits to site administrators for creating an internal network structure suiting each administrator's needs.

### ***IPv6 Implementation of the Unicast Address Space***

In theory, any size for  $n$  and  $m$  (see Table 25-2) could be used. The implementation chosen for IPv6, however, assigns 48 bits to the routing prefix and 16 bits to the subnet identifier. This means 64 bits are available for interface identifiers, which are constructed based on the IEEE EUI-64 format, as described in the next section. Thus, the overall IPv6 unicast address format is constructed as shown in Table 25-3 and illustrated in Figure 25-3.

**Table 25-3:** IPv6 Global Unicast Address Format

Field Name	Size (Bits)	Description
Prefix	48	Global Routing Prefix: The network ID or prefix of the address that's used for routing. The first three bits are 001 to indicate a unicast address.
Subnet ID	16	Subnet Identifier: A number that identifies a subnet within the site.
Interface ID	64	Interface ID: The unique identifier for a particular interface (host or other device). It is unique within the specific prefix and subnet.



**Figure 25-3:** IPv6 global unicast address format

**KEY CONCEPT** The part of the IPv6 address space set aside for unicast addresses is structured into an address format that uses the first 48 bits for the *routing prefix* (like a network ID), the next 16 bits for a *subnet ID*, and the final 64 bits for an *interface ID* (like a host ID).

Due to this structure, most end sites (regular companies and organizations, as opposed to ISPs) will be assigned IPv6 networks with a 48-bit prefix. In common parlance, these network identifiers have now come to be called *48s* or /48s.

The 16 bits of subnet ID allow each site considerable flexibility in creating subnets that reflect the site's network structure. Here are some example uses of the 16 bits:

- A smaller organization can just set all the bits in the subnet ID to zero and have a flat internal structure.
- A medium-sized organization could use all the bits in the subnet ID to perform the equivalent of straight subnetting under IPv4, thereby assigning a different subnet ID to each subnet. There are 16 bits here, and this allows a whopping 65,536 subnets!
- A larger organization can use the bits to create a multiple-level hierarchy of subnets, exactly like IPv4's Variable Length Subnet Masking (VLSM). For example, the company could use two bits to create four subnets. It could then take the next three bits to create eight sub-subnets in some or all of the four subnets. There would still be 11 more bits to create sub-sub-subnets, and so forth.

### **Original Division of the Global Routing Prefix: Aggregators**

The global routing prefix is similarly divided into a hierarchy, but one that has been designed for the use of the entire Internet, like CIDR. There are 45 bits available here (48 bits minus the first three that are fixed at 001). That is a lot. When the unicast address structure was first detailed in RFC 2374, that document described a specific division of the 45 bits based on a two-level hierarchical topology of Internet registries and providers. These organizations were described as follows:

**Top-Level Aggregators (TLAs)** These refer to the largest Internet organizations, which were to be assigned large blocks of IPv6 addresses from registration authorities.

**Next-Level Aggregators (NLAs)** These organizations would get blocks of addresses from TLAs and divide them for end-user organizations (sites).

The 45 bits were split between these two uses, with a few bits reserved in the middle to allow expansion of either field if needed. Thus, the RFC 2374 structure for the 45 bits appeared as listed in Table 25-4.

**Table 25-4:** Historical IPv6 Unicast Routing Prefix Structure

Field Name	Size (Bits)	Description
TLA ID	13	Top-Level Aggregation (TLA) Identifier: A globally unique identifier for the top-level aggregator. There are 13 bits, so there were a maximum of 8,192 TLAs allowed.
RES	8	Reserved: These 8 bits were reserved for future use and set to zero. By leaving these 8 bits between the TLA ID and NLA ID unused, they could later be used to expand either the TLA ID or NLA ID fields as needed.
NLA ID	24	Next-Level Aggregation (NLA) Identifier: Each TLA was given this 24-bit field to generate blocks of addresses for allocation to its customers. The NLA ID is unique for each TLA ID. The use of the 24 bits was left up to the TLA organization.

You'll notice my use of the past tense in the description of the TLA/NLA structure, and that table heading is a pretty big giveaway, too. In August 2003, RFC 3587 was published, which in a nutshell says, "Uh, never mind about all that TLA/NLA stuff." The decision was made that having this structure hardwired into an Internet standard was inflexible, and it made more sense to let the regional Internet registries (APNIC, ARIN, LACNIC, and RIPE) decide for themselves how to use the 45 bits.

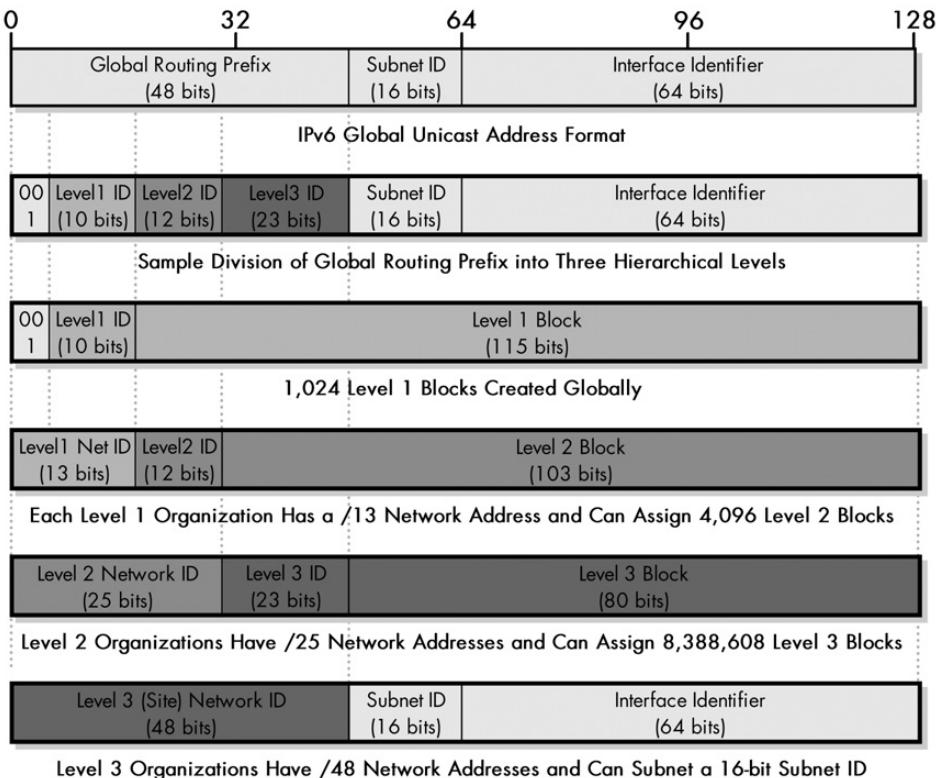
**NOTE** *The obsoleting of the TLA/NLA structure occurred after many years of people getting used to it, so for some time to come, you will still routinely see those terms mentioned in IPv6 descriptions. (This is why I included discussion of them here.)*

### A Sample Division of the Global Routing Prefix into Levels

There is no single structure for determining how the 48-bit routing prefix is divided in the global unicast hierarchy. As one example, it might be possible to divide it into three levels, as shown in Table 25-5 and illustrated in Figure 25-4.

**Table 25-5:** Example IPv6 Unicast Routing Prefix Structure

Field Name	Size (Bits)	Description
(Unicast Indicator)	3	Each unicast address starts with 001; there is no official name for this (it used to be called the <i>format prefix</i> ).
Level1 ID	10	Level 1 Identifier: The identifier of the highest level in the hierarchy. This would be used for assigning the largest blocks of addresses in the global hierarchy to the biggest Internet organizations. The number of level 1 organizations would be 210, or 1,024.
Level2 ID	12	Level 2 Identifier: Each block assigned to a level 1 organization would use 12 bits to create 4,096 address blocks to divide among the lower-level organizations it serves.
Level3 ID	23	Level 3 Identifier: Each level 2 organization has 23 bits to use to divide its level 2 address block. Thus, it could create over 8 million individual /48 address blocks to assign to end-user sites. Alternatively, the 23 bits could be divided further into still lower levels to reflect the structure of the level 2 organization's customers.



**Figure 25-4: Example of IPv6 unicast routing prefix structure** The top row shows the global IPv6 unicast address format. The second shows one way to divide the global routing prefix into three levels using 10, 12, and 23 bits, respectively. The third row shows how the first 10 bits are used to create  $2^{10}$ , or 1,024, different level 1 blocks. The next row illustrates that for each of these 13-bit prefixes, you could have  $2^{12}$ , or 4,096, level 2 blocks. Then, within each 25-bit level 2 ID, you have 23 bits, or 8,388,608, level 3 blocks. At the bottom, a level 3 or /48 would be assigned to an individual organization.

This is just one possible theoretical way that the bits in a /48 network address could be assigned. As you can see, with so many bits, there is a lot of flexibility. In the previous scheme, you can have over four million level 2 organizations, each of which can assign eight million /48 addresses. And each of those is equivalent in size to an IPv4 Class B address (over 65,000 hosts)!

The removal of RFC 2374's fixed structure for the global routing prefix is consistent with the IPv6 development team's efforts to emphasize that bit fields and structures are used only for allocating addresses and not for routing purposes. The addresses themselves, once created, are not interpreted by hardware on an internetwork based on this format. To routers, the only structure that matters is the division between the network ID and host ID, given by the prefix length that trails the IP address, and this division can occur at any bit boundary. These hardware devices just see 128 bits of an IP address and use it without any knowledge of hierarchical address divisions or levels.

Incidentally, the key to obtaining the allocation benefits of the aggregatable unicast address format is the abundance of bits available to us under IPv6. The ability to have these hierarchical levels while still allowing 64 bits for the interface identifier is one of the main reasons why IPv6 designers went all the way from 32 bits to 128 bits for address size. By creating this structure, we maintain flexibility, while avoiding the potential chaos of trying to allocate many different network sizes within the 128 bits.

Note that anycast addresses are structured in the same way as unicast addresses, so they are allocated according to this same model. (Multicast addresses are not.)

## IPv6 Interface Identifiers and Physical Address Mapping

In IPv4, IP addresses have no relationship to the addresses used for underlying data link layer network technologies. A host that connects to a TCP/IP network using an Ethernet network interface card (NIC) has an Ethernet MAC address and an IP address, but the two numbers are distinct and unrelated in any way. IP addresses are assigned manually by administrators without any regard for the underlying physical address.

With the overhaul of addressing in IPv6, an opportunity presented itself to create a better way of mapping IP unicast addresses and physical network addresses. Implementing this superior mapping technique was one of the reasons why IPv6 addresses were made so large. With 128 total bits, even with a full 45 bits reserved for the network prefix and 16 bits for the site subnet, we are still left with 64 bits to use for the *interface identifier (interface ID)*, which is analogous to the host ID under IPv4.

Having so many bits at our disposal gives us great flexibility. Instead of using arbitrary, made-up identifiers for hosts, we can base the interface ID on the underlying data link layer hardware address, as long as that address is no greater than 64 bits in length. Since virtually all devices use layer 2 addresses of 64 bits or fewer, there is no problem in using those addresses for the interface ID in IP addresses. This provides an immediate benefit: It makes networks easier to administer, since we don't need to record two arbitrary numbers for each host. The IP address can be derived from the MAC address and the network ID. It also means that we can tell the IP address from the MAC address and vice versa.

The actual mapping from data link layer addresses to IP interface IDs depends on the particular technology. It is essential that all devices on the same network use the same mapping technique, of course. By far, the most common type of layer 2 addresses in networking are IEEE 802 MAC addresses, which are used by Ethernet and other IEEE 802 Project networking technologies. These addresses have 48 bits, arranged into two blocks of 24. The upper 24 bits are arranged into a block called the *organizationally unique identifier (OUI)*, with different values assigned to individual organizations. The lower 24 bits are then used for an identifier for each specific device.

The IEEE has also defined a format called the *64-bit extended unique identifier*, which is abbreviated *EUI-64*. It is similar to the 48-bit MAC format, except that while the OUI remains at 24 bits, the device identifier becomes 40 bits instead of 24. This gives each manufacturer 65,536 times as many device addresses within its OUI.

A form of this format, called *modified EUI-64*, has been adopted for IPv6 interface IDs. To get the modified EUI-64 interface ID for a device, you simply take the

EUI-64 address and change the seventh bit from the left (the universal/local, or U/L, bit) from a 0 to a 1.

Of course, most devices still use the older 48-bit MAC address format. These can be converted to EUI-64 and then modified to EUI-64 form for creating an IPv6 interface ID. The process is as follows:

1. Take the 24-bit OUI portion, the leftmost 24 bits of the Ethernet address, and put them into the leftmost 24 bits of the interface ID. Take the 24-bit local portion (the rightmost 24 bits of the Ethernet address) and put it into the rightmost 24 bits of the interface ID.
2. In the remaining 16 bits in the middle of the interface ID, put the value 11111111 11111110, FFFE in hexadecimal.
3. The address is now in EUI-64 form. Change the universal/local bit (bit 7 from the left, shown in bold in Figure 25-5) from a 0 to a 1. This gives the modified EUI-64 interface ID.

**KEY CONCEPT** The last 64 bits of IPv6 unicast addresses are used for interface IDs, which are created in a special format called *modified EUI-64*. A simple process can be used to determine the interface ID from the 48-bit MAC address of a device like an Ethernet network interface card. This can then be combined with a network prefix (routing prefix and subnet ID) to determine a corresponding IPv6 address for the device.

Let's take as an example the Ethernet address of 39-A7-94-07-CB-D0. Here are the steps for conversion (illustrated in Figure 25-5):

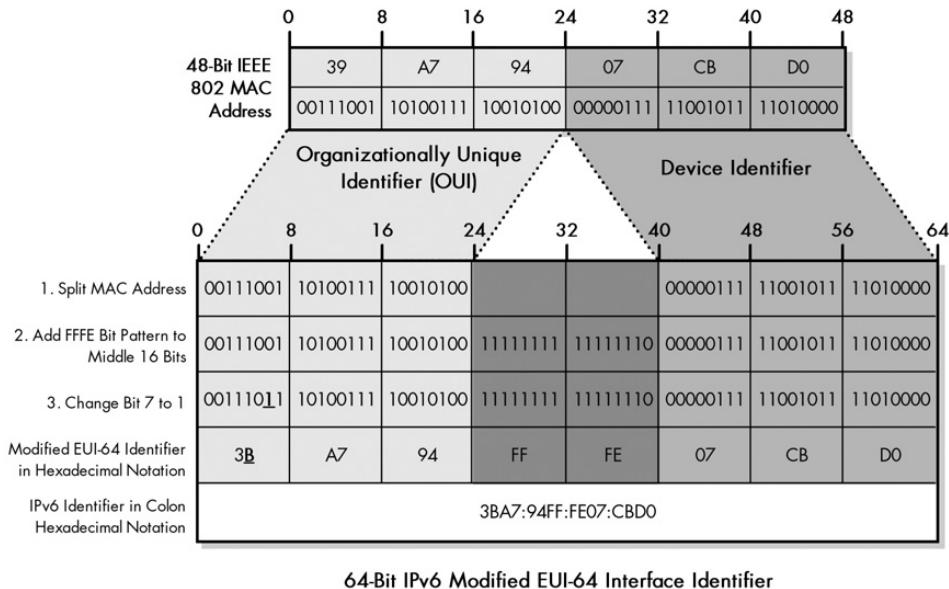
1. Take 39-A7-94, the first 24 bits of the identifier, and put it into the first (leftmost) 24 bits of the address. The local portion of 07-CB-D0 becomes the last 24 bits of the identifier.
2. The middle 16 bits are given the value FF-FE.
3. Change the seventh bit from 0 to 1, which changes the first octet from 39 to 3B.

The identifier thus becomes 3B-A7-94-FF-FE-07-CB-D0, or in IPv6 colon hexadecimal notation, 3BA7:94FF:FE07:CBD0. The first 64 bits of the device's address are supplied using the global unicast address format.

The only drawback of this technique is that if the physical hardware changes, so does the IPv6 address.

## IPv6 Special Addresses: Reserved, Private, Unspecified, and Loopback

Just as certain IPv4 address ranges are designated for reserved, private, and other unusual addresses, a small part of the monstrous IPv6 address space has been set aside for special addresses. The purpose of these addresses and address blocks is to provide addresses for special requirements and private use in IPv6 networks. Since even relatively small pieces of IPv6 are still enormous, setting aside 0.1 percent of the address space for a particular use still generally yields more addresses than anyone will ever need.



**Figure 25-5:** Converting IEEE 802 MAC addresses to IPv6 modified EUI-64 identifiers

## Special Address Types

There are four basic types of special IPv6 addresses:

**Reserved Addresses** A portion of the address space is set aside as reserved for various uses by the IETF, both present and future. Unlike IPv4, which has many small reserved blocks in various locations in the address space, the reserved block in IPv6 is at the “top” of the address space, beginning with 0000 0000 (or 00 for the first hexadecimal octet). This represents 1/256th of the total address space. Some of the special addresses you’ll see shortly come from this block. IPv4 address embedding is also done within this reserved address area.

**NOTE** Reserved addresses are not the same as unassigned addresses. The latter term just refers to blocks whose use has not yet been determined.

**Private/Unregistered/Nonroutable Addresses** A block of addresses is set aside for private addresses, just as in IPv4, except that like everything in IPv6 the private address block in IPv6 is much larger. These private addresses are local only to a particular link or site and, therefore, are never routed outside a particular company’s network. Private addresses are indicated by the address having “1111 1110 1” for the first nine bits. Thus, private addresses have a first octet value of FE in hexadecimal, with the next hexadecimal digit being from 8 to F. These addresses are further divided into two types based on their scope: site-local and link-local, as discussed shortly.

**Loopback Address** Like IPv4, a provision has been made for a special loopback address for testing; datagrams sent to this address “loop back” to the sending device. However, in IPv6, there is just one address for this function, not a whole block (which was never needed in the first place). The loopback address is 0:0:0:0:0:0:1, which is normally expressed using zero compression as ::1.

**Unspecified Address** In IPv4, an IP address of all zeros has a special meaning: It refers to the host itself and is used when a device doesn’t know its own address. In IPv6, this concept has been formalized, and the all-zeros address (0:0:0:0:0:0) is named the *unspecified address*. It is typically used in the source field of a datagram sent by a device seeking to have its IP address configured. Zero compression can be applied to this address; since it is all zeros, the address becomes just ::. (I consider this confusing, myself. I think something like 0::0 is a lot clearer and short enough.)

**KEY CONCEPT** In IPv6, a special *loopback address*, 0:0:0:0:0:0:1 (::1 in compressed form) is set aside for testing purposes. The *unspecified address*, 0:0:0:0:0:0 (:: in compressed form) is used to indicate an unknown address. A block of *private* or *local* addresses is defined. This block is the set of all addresses beginning with 1111 1110 1 as the first nine bits.

## IPv6 Private Addresses Type Scopes

Now let’s take a closer look at private addresses. In IPv6, these are called *local-use* addresses, with the name conveying clearly what they are for. They are also sometimes called *link-layer* addresses. You’ll recall that IPv4 private addresses were commonly used when public addresses could not be obtained for all devices, sometimes in combination with technologies like Network Address Translation (NAT). In IPv6, trickery like NAT isn’t required. Instead, local-use addresses are intended for communication that is inherently designed to be sent to local devices only. For example, neighbor discovery functions using the IPv6 Neighbor Discovery (ND) protocol employ local-use addresses.

The *scope* of local addresses is obviously a local network, not the global scope of public Internet addresses. Local addresses in IPv6 are further divided into two types, reflecting a division of local scope:

**Site-Local Addresses** These addresses have the scope of an entire site or organization. They allow addressing within an organization without having to use a public prefix. Routers will forward datagrams using site-local addresses within the site, but not addresses outside it to the public Internet. Site-local addresses are differentiated from link-local addresses by having a tenth bit of 1 following the nine starting address bits that are common to all private IPv6 addresses. Thus, they begin with 1111 1110 11. In hexadecimal, site-local addresses begin with FE, and then C to F for the third digit. So, these addresses start with FEC, FED, FEE, or FEF.

**Link-Local Addresses** These addresses have a smaller scope than site-local addresses; they refer only to a particular physical link (physical network). Routers will not forward datagrams using link-local addresses at all—not even within the organization. These addresses are only for local communication on a particular physical network segment. They can be used for address configuration or for ND

functions such as address resolution and ND. Link-local addresses are differentiated from site-local addresses by having a tenth bit of 0 following the nine initial address bits common to all private IPv6 addresses: 1111 1110 1. Thus, site-local addresses begin with FE, and then 8 to B for the third hexadecimal digit. So, these addresses start with FE8, FE9, FEA, or FEB.

**KEY CONCEPT** IPv6 site-local addresses allow data to be sent only to the devices within a site or organization. They begin with FEC, FED, FEE, or FEF in hexadecimal. IPv6 link-local addresses are used only on a particular local link (physical network), typically for special purposes such as address resolution or Neighbor Discovery (ND). They start with FE8, FE9, FEA, or FEB.

Note that site-local IPv6 addresses are the equivalent of IPv4 private addresses, since they are routed throughout the organization. The concept of link-local scope is new to IPv6.

## IPv6/IPv4 Address Embedding

Due to the importance of IP and the significance of the changes made in IPv6, deployment of the newer version of the protocol will not occur all at once. A *transition* from IPv4 to IPv6 will be required. This transition requires careful planning. It is anticipated that the migration from IPv4 to IPv6 will take many years, as I mentioned earlier.

IPv6 is backward-compatible with IPv4 provided that you use special techniques. For example, to enable communication between islands of IPv6 devices connected by IPv4 networks, you may need to employ tunneling. To support IPv4/IPv6 compatibility, a scheme was developed to allow IPv4 addresses to be *embedded* within the IPv6 address structure. This method takes regular IPv4 addresses and puts them in a special IPv6 format, so that they are recognized as being IPv4 addresses by certain IPv6 devices.

Since the IPv6 address space is so much bigger than the one in IPv4, embedding the latter within the former is easy—it's like tucking a compact sedan into the hold of a cargo ship! The embedding address space is part of the reserved address block whose addresses begin with eight 0 bits, but it's only a relatively small part. Two different embedding formats are used to indicate the capabilities of the device that's using the embedded address:

**IPv4-Compatible IPv6 Addresses** These are special addresses assigned to IPv6-capable devices, such as *dual-stack* devices that use both IPv4 and IPv6. They have all zeros for the middle 16 bits; thus, they start off with a string of 96 zeros, followed by the IPv4 address. An example of such an address would be 0:0:0:0:0:101.45.75.219 in mixed notation, or more succinctly, ::101.45.75.219. Figure 25-6 illustrates IPv4-compatible IPv6 representation.

**IPv4-Mapped IPv6 Addresses** These are regular IPv4 addresses that have been mapped into the IPv6 address space. They are used for devices that are IPv4-capable only. They have a set of 16 ones after the initial string of 80 zeros and then the

IPv4 address. So if an IPv4 device has the address 222.1.41.90, such as the one shown in Figure 25-7, it would be represented as 0:0:0:0:FFFF:222.1.41.90, or ::FFFF:222.1.41.90.

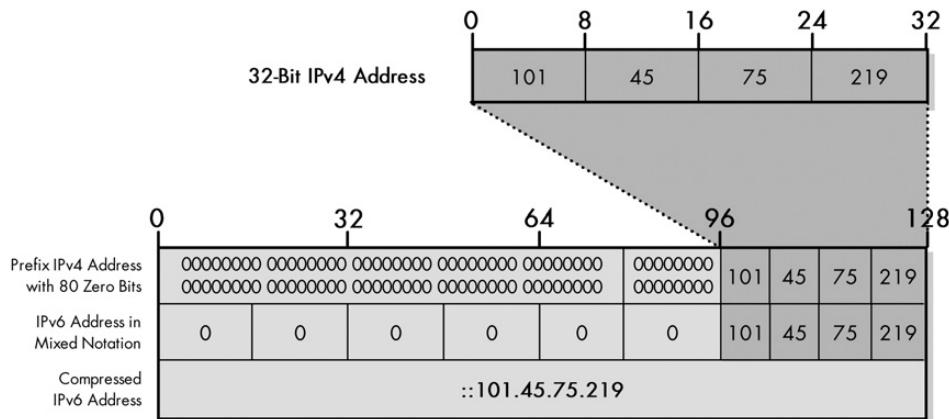


Figure 25-6: IPv4-compatible embedded IPv6 address representation

The difference between these two is subtle but important. Both have zeros for the first 80 bits of the address and put the embedded IPv4 address into the last 32 bits of the IPv6 address format. They differ in the value of the 16 remaining bits in between (bits 81 to 96, counting from the left). IPv4-compatible IPv6 addresses are used only for devices that are actually IPv6-aware; the IPv4-compatible address is in addition to its conventional IPv6 address. In contrast, if the FFFF is seen for the 16 bits after the initial 80, this designates a conventional IPv4 devices whose IPv4 address has been mapped into the IPv6 format. It is not an IPv6-capable device.

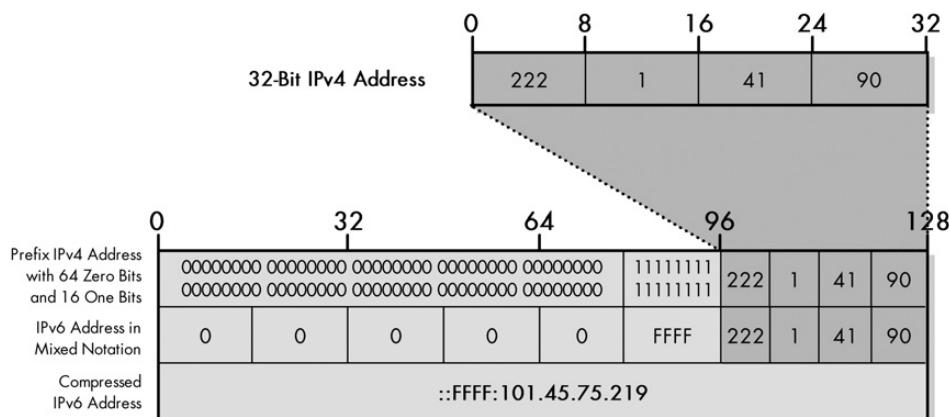


Figure 25-7: IPv4-mapped embedded IPv6 address representation

**KEY CONCEPT** *IPv4 address embedding* is used to create a relationship between an IPv4 address and an IPv6 address to help you transition from IPv4 to IPv6. One type, the *IPv4-compatible IPv6 address*, is used for devices that are compatible with both IPv4 and IPv6; it begins with 96 zero bits. The other, the *IPv4-mapped address*, is used for mapping IPv4 devices that are not compatible with IPv6 into the IPv6 address space; it begins with 80 zeros followed by 16 ones.

## IPv6 Multicast and Anycast Addressing

One of the most significant modifications in the general addressing model in IPv6 was a change to the basic types of addresses and how they were used. Unicast addresses are still the choice for the vast majority of communications as in IPv4, but the “bulk” addressing methods are different in IPv6. Broadcast as a specific addressing type has been eliminated. Instead, support for multicast addressing has been expanded and made a required part of the protocol, and a new type of addressing called *anycast* has been implemented.

### IPv6 Multicast Addresses

Let’s start by looking at multicast under IPv6. Multicasting is used to allow a single device to send a datagram to a group of recipients. IPv4 supported multicast addressing using the Class D address block in the classful addressing scheme (see Chapter 17). Under IPv6, multicast addresses are allocated from the multicast block. This is 1/256th of the address space, and it consists of all addresses that begin with 1111 1111. Thus, any address starting with FF in colon hexadecimal notation is an IPv6 multicast address.

The remaining 120 bits of address space are enough to allow the definition of, well, a gazillion or three multicast addresses. (OK, it’s officially about 1.3 trillion trillion addresses.) The allocation of unicast addresses was organized by using a special format to divide these many bits, and the same thing was done for multicast addresses. The format for multicast addresses is explained in Table 25-6 and illustrated in Figure 25-8.

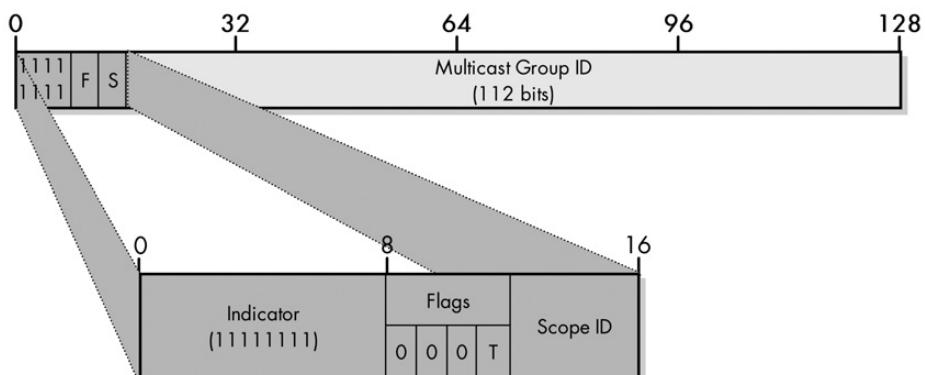


Figure 25-8: IPv6 multicast address format

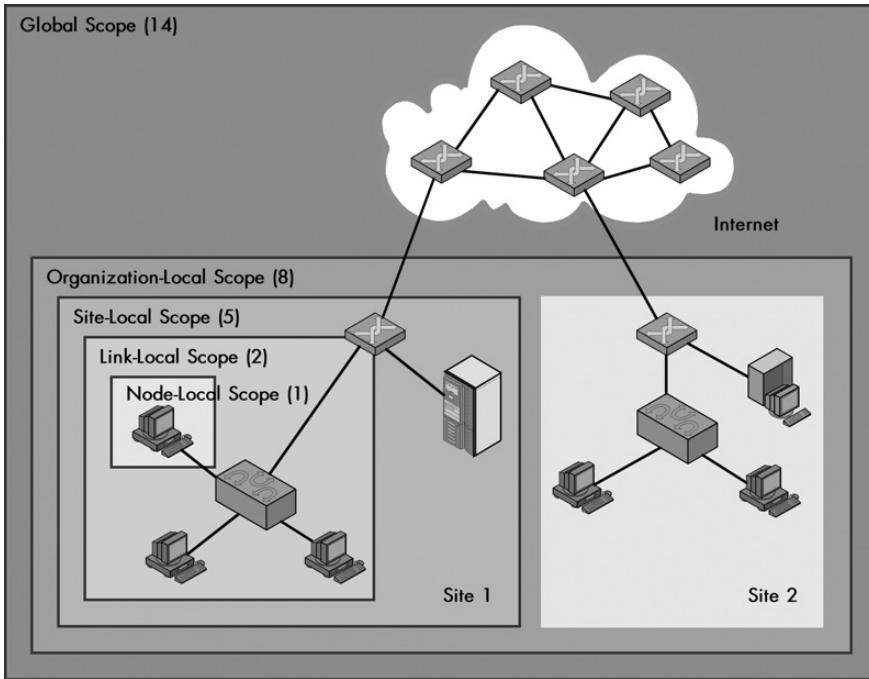
**Table 25-6:** IPv6 Multicast Address Format

Field Name	Size (Bits)	Description
(Indicator)	8	The first eight bits are always 1111 1111, which indicates a multicast address. This used to be called the <i>format prefix</i> before the term was dropped (as explained in the section about IPv6 address space allocation earlier in this chapter). The field now has no name.
Flags	4	Four bits are reserved for flags that can be used to indicate the nature of certain multicast addresses. Currently, the first three of these are unused and set to zero. The fourth is the T (Transient) flag. If left as zero, this marks the multicast address as a permanently assigned, well-known multicast address, as you will see shortly. If set to one, this means this is a <i>transient</i> multicast address, meaning that it is not permanently assigned.
Scope ID	4	These four bits are used to define the scope of the multicast address; 16 different values from 0 to 15 are possible. This field allows creation of multicast addresses that are global to the entire Internet, or restricted to smaller spheres of influence such as a specific organization, site, or link. The currently defined values (in decimal) are as follows: 0 = Reserved 1 = Node-Local Scope 2 = Link-Local Scope 5 = Site-Local Scope 8 = Organization-Local Scope 14 = Global Scope 15 = Reserved
Group ID	112	Defines a particular group within each scope level.

### Multicast Scopes

The notion of explicitly scoping multicast addresses is important. Globally scoped multicast addresses must be unique across the entire Internet, but locally scoped addresses are unique only within the organization. This provides tremendous flexibility, as every type of multicast address actually comes in several versions: one that multicasts only within a node, one that multicasts on the local link (local network), one that multicasts on the local site, and so on. The scope also allows routers to immediately determine how broadly they should propagate multicast datagrams in order to improve efficiency and eliminate problems with traffic being sent outside the area for which it is intended. Figure 25-9 illustrates the notion of multicast scope graphically.

**KEY CONCEPT** Multicast addresses are used to send data to a number of devices on an internetwork simultaneously. In IPv6, each multicast address can be specified for a variety of different scopes, thereby allowing a transmission to be targeted to either a wide or a narrow audience of recipient devices.



**Figure 25-9: IPv6 multicast scope** This diagram shows how the notion of scope allows IPv6 multicasts to be limited to specific spheres of influence. The tightest scope is node-local scope, with a scope ID value of 1. As the scope ID value increases, the scope expands to cover the local network, site, organization, and finally, entire Internet.

### Well-Known Multicast Addresses

The Transient flag allows for the explicit determination of which multicast addresses are available for normal use compared to which ones are set aside as well known. Several well-known multicast addresses are defined by setting aside certain group IDs that are used for a number of different scope ID values. Table 25-7 shows these values; the  $x$  in the multicast address pattern is the hexadecimal digit corresponding to the four-bit scope ID field.

The all-nodes and all-routers multicast addresses enable the equivalent function of what broadcast used to perform in IPv4. Again, the concept of scope is important in a multicast of this type, because we don't want to try to send a message to all nodes on the global Internet, for example. So when the all-routers address is used with a scope value of 2, it means "all routers on the local link." If it is used with a value of 5, it means "all routers in this site."

### Solicited-Node Multicast Addresses

In addition to the regular multicast addresses, each unicast address has a special multicast address called its *solicited-node address*. This address is created through a special mapping from the device's unicast address. Solicited-node addresses are used by the IPv6 ND protocol (see Chapter 36) to provide more efficient address resolution than the Address Resolution Protocol (ARP; see Chapter 13) technique used in IPv4.

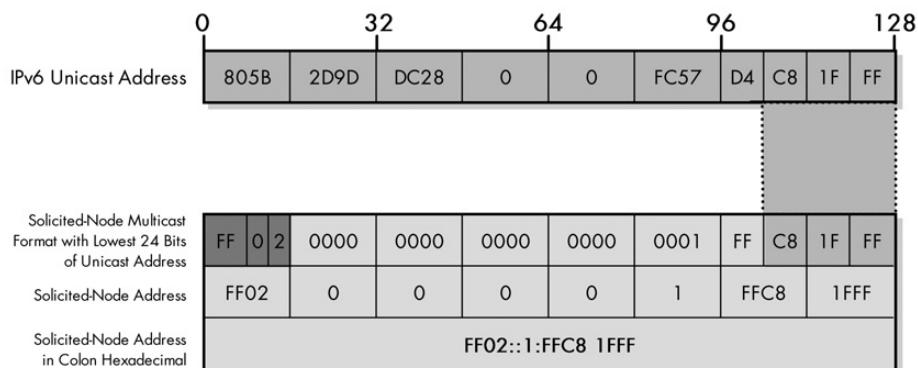
**Table 25-7: Important IPv6 Well-Known Multicast Addresses**

Multicast Address Pattern	Valid Scope Values (Decimal)	Designation	Description
FF0x:0:0:0:0:0	0 to 15	Reserved	All multicast addresses where the 112-bit group ID is zero are reserved.
FF0x:0:0:0:0:1	1, 2	All Nodes	When the group ID is equal to exactly 1, this is a multicast to all nodes. Both node-local (FF01:0:0:0:0:1) and link-local (FF02:0:0:0:0:1) all-nodes multicast addresses are possible.
FF0x:0:0:0:0:2	1, 2, 5	All Routers	When the group ID is equal to exactly 2, this designates all routers within a specific scope as the recipients. Valid scope values are node-local, link-local, and site-local.

All solicited-node addresses have their T flag set to zero and a scope ID of 2, so they start with FF02. The 112-bit group ID is broken down as follows (see Figure 25-10):

- Eighty bits consisting of 79 zeros followed by a single one. This means that the next five hexadecimal values are 0000:0000:0000:0000:0001 in colon hexadecimal notation, or more succinctly, 0:0:0:0:1.
- Eight ones: FF.
- Twenty-four bits taken from the bottom 24 bits of its unicast address.

So, these addresses start with FF02:0:0:0:1:FF, followed by the bottom 24 bits of the unicast address. Thus, the node with IP address 805B:2D9D:DC28:0:0:FC57:D4C8:1FFF would have a solicited-node address of FF02:0:0:0:1:FFC8:1FFF (or FF02::1:FFC8:1FFF).



**Figure 25-10: IPv6 solicited-node address calculation** The solicited-node multicast address is calculated from a unicast address by taking the last 24 bits of the address and prepending them with the IPv6 partial address FF02:0:0:0:0:1:FF. This shows the example address from Figure 25-2 converted to its solicited-node address, FF02::1:FFC8:1FFF.

**KEY CONCEPT** Each unicast address has an equivalent *solicited-node multicast address* that is created from the unicast address and used when other devices need to reach it on the local network.

## **IPv6 Anycast Addresses**

Anycast addresses are a unique type of address that is new to IP in IPv6. The IPv6 implementation is based on the material in RFC 1546, “Host Anycasting Service.” Anycast addresses can be considered a conceptual cross between unicast and multicast addressing. Where unicast says, “Send this to one address,” and multicast says, “Send this to every member of this group,” anycast says, “Send this to any one member of this group.” Naturally, in choosing which member to send to, we would, for efficiency, normally send to the closest one—that is, the closest in routing terms. So, we can normally also consider *anycast* to mean, “Send this to the closest member of this group.”

The idea behind anycast is to enable functionality that was previously difficult to implement in TCP/IP. Anycast was specifically intended to provide flexibility in situations where we need a service that is provided by a number of different servers or routers but don’t really care which one provides it. In routing, anycast allows datagrams to be sent to whichever router in a group of equivalent routers is closest, and to allow load sharing among routers and dynamic flexibility if certain routers go out of service. Datagrams sent to the anycast address will automatically be delivered to the device that is easiest to reach.

Perhaps surprisingly, there is no special anycast-addressing scheme. Anycast addresses are the same as unicast addresses. An anycast address is created automatically when a unicast address is assigned to more than one interface.

**KEY CONCEPT** Anycast addresses are new in IPv6 and can be used to set up a group of devices, any one of which can respond to a request sent to a single IP address.

Like multicast, anycast creates more work for routers, because it is more complicated than unicast addressing. In particular, the further apart the devices that share the anycast address are, the more complexity. Anycasting across the global Internet would be potentially difficult to implement, and IPv6 anycasting was designed for devices that are proximate to each other, generally within the same network. Also, at present, due to the Internet community’s relative inexperience with anycast, only routers, not individual hosts, use anycast addresses.

## **IPv6 Autoconfiguration and Renumbering**

One of the most interesting and potentially valuable addressing features implemented in IPv6 is a facility that allows devices on an IPv6 network to actually configure themselves independently. In IPv4, hosts were originally configured manually. Later, host configuration protocols like the Dynamic Host Configuration Protocol (DHCP; see Chapter 61) enabled servers to allocate IP addresses to hosts that joined the network. IPv6 takes this a step further by defining a method for some devices to automatically configure their IP address and other parameters without the need for a server. It also defines a method whereby the IP addresses on a network can be renumbered (changed en masse). These are the sorts of features that make TCP/IP network administrators drool.

The IPv6 autoconfiguration and renumbering feature is defined in RFC 2462, “IPv6 Stateless Address Autoconfiguration.” The word *stateless* contrasts this method to the server-based method using something like DHCPv6, which is called *stateful*. (This word, like *classful*, makes me cringe.) This method is called stateless because it begins with no information (or *state*) at all for the host to work with. It has no need for a DHCP server.

## **IPv6 Stateless Autoconfiguration**

Stateless autoconfiguration exploits several other new features in IPv6, including link-local addresses, multicasting, the ND protocol, and the ability to generate the interface ID of an address from the underlying data link layer address. The general idea is to have a device generate a temporary address until it can determine the characteristics of the network it is on, and then create a permanent address it can use based on that information. In the case of multihomed devices, autoconfiguration is performed for each interface separately.

The following is a summary of the steps a device takes when using stateless autoconfiguration:

1. **Link-Local Address Generation** The device generates a link-local address. You’ll recall that this is one of the two types of local-use IPv6 addresses. Link-local addresses have 1111 1110 10 for the first 10 bits. The generated address uses those 10 bits, followed by 54 zeros and then the 64-bit interface ID. Typically, this will be derived from the data link layer (MAC) address as explained in the “IPv6 Interface Identifiers and Physical Address Mapping” section earlier in this chapter, or it may be a “token” generated in some other manner.
2. **Link-Local Address Uniqueness Test** The node tests to ensure that the address it generated isn’t already in use on the local network. (This is very unlikely to be an issue if the link-local address came from a MAC address; it is more likely that the address is already in use if it was based on a generated token.) It sends a Neighbor Solicitation message using the ND protocol. In response, it listens for a Neighbor Advertisement, which indicates that another device is already using its link-local address. If so, either a new address must be generated or autoconfiguration fails, and another method must be employed.
3. **Link-Local Address Assignment** Assuming the uniqueness test passes, the device assigns the link-local address to its IP interface. This address can be used for communication on the local network, but not on the wider Internet (since link-local addresses are not routed).
4. **Router Contact** The node next attempts to contact a local router for more information on continuing the configuration. This is done either by listening for Router Advertisement messages sent periodically by routers or by sending a specific Router Solicitation message to ask a router for information on what to do next. This process is described in the section on the IPv6 ND protocol, in Chapter 36.

5. **Router Direction** The router provides direction to the node about how to proceed with the autoconfiguration. It may tell the node that on this network stateful autoconfiguration is in use, and it may give it the address of a DHCP server to use. Alternatively, it will tell the host how to determine its global Internet address.
6. **Global Address Configuration** Assuming that stateless autoconfiguration is in use on the network, the host will configure itself with its globally unique Internet address. This address is generally formed from a network prefix provided to the host by the router. The prefix is combined with the device's identifier, as generated in step 1.

Clearly, this method has numerous advantages over both manual and server-based configuration. It is particularly helpful in supporting the mobility of IP devices, because they can move to new networks and get a valid address without any knowledge of local servers or network prefixes. At the same time, it still allows for the management of IP addresses using the (IPv6-compatible) version of DHCP, if that is desired. Routers on the local network will typically tell hosts which type of autoconfiguration is supported using special flags in Internet Control Message Protocol version 6 (ICMPv6) Router Advertisement messages (see Chapter 35).

**KEY CONCEPT** IPv6 includes an interesting feature called *stateless address autoconfiguration*, which allows a host to actually determine its own IPv6 address from its layer 2 address by following a special procedure.

## **IPv6 Device Renumbering**

The renumbering of devices is a method related to autoconfiguration. Like host configuration, it can be implemented using protocols like DHCP through the use of IP address leases that expire after a period of time. Under IPv6, networks can be renumbered by having routers specify an expiration interval for network prefixes when autoconfiguration is done. Later, they can send a new prefix to tell devices to regenerate their IP addresses. Devices can actually maintain the old deprecated address for a while, and then move over to the new address.

RFC 2894 defined a similar technique for renumbering router addresses. It uses special ICMPv6 messages and is described in Chapter 35.