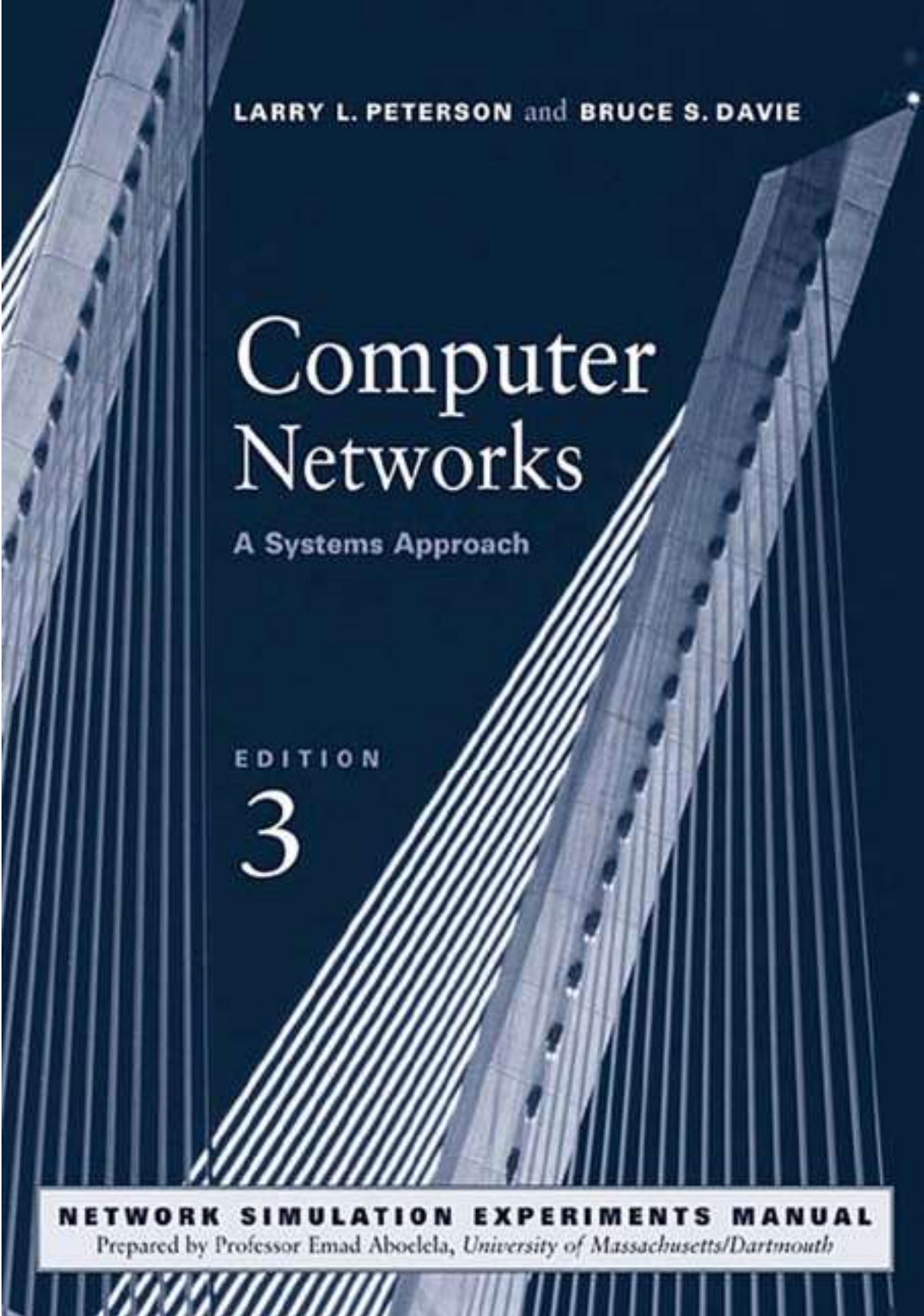




LARRY L. PETERSON and BRUCE S. DAVIE

Computer Networks

A Systems Approach

EDITION

3

NETWORK SIMULATION EXPERIMENTS MANUAL

Prepared by Professor Emad Aboelela, *University of Massachusetts/Dartmouth*

THIRD EDITION

COMPUTER NETWORKS

A Systems Approach

This book belongs to Janvi Shah (jshah@sughrue.com)

Copyright Elsevier 2023

The Morgan Kaufmann Series in Networking

Series Editor, David Clark, M.I.T.

- Computer Networks: A Systems Approach, 3e*
Larry L. Peterson and Bruce S. Davie
- Network Architecture, Analysis, and Design, 2e*
James D. McCabe
- MPLS Network Management: MIBs, Tools, and Techniques*
Thomas D. Nadeau
- Developing IP-Based Services: Solutions for Service Providers and Vendors*
Monique Morrow and Kateel Vijayananda
- Telecommunications Law in the Internet Age*
Sharon K. Black
- Optical Networks: A Practical Perspective, 2e*
Rajiv Ramaswami and Kumar N. Sivarajan
- Internet QoS: Architectures and Mechanisms*
Zheng Wang
- TCP/IP Sockets in Java: Practical Guide for Programmers*
Michael J. Donahoo and Kenneth L. Calvert
- TCP/IP Sockets in C: Practical Guide for Programmers*
Kenneth L. Calvert and Michael J. Donahoo
- Multicast Communication: Protocols, Programming, and Applications*
Ralph Wittmann and Martina Zitterbart
- MPLS: Technology and Applications*
Bruce Davie and Yakov Rekhter
- High-Performance Communication Networks, 2e*
Jean Walrand and Pravin Varaiya
- Internetworking Multimedia*
Jon Crowcroft, Mark Handley, and Ian Wakeman
- Understanding Networked Applications: A First Course*
David G. Messerschmitt
- Integrated Management of Networked Systems: Concepts, Architectures, and their Operational Application*
Heinz-Gerd Hegering, Sebastian Abeck, and Bernhard Neumair
- Virtual Private Networks: Making the Right Connection*
Dennis Fowler
- Networked Applications: A Guide to the New Computing Infrastructure*
David G. Messerschmitt
- Modern Cable Television Technology: Video, Voice, and Data Communications*
Walter Ciciora, James Farmer, and David Large
- Switching in IP Networks: IP Switching, Tag Switching, and Related Technologies*
Bruce S. Davie, Paul Doolan, and Yakov Rekhter
- Wide Area Network Design: Concepts and Tools for Optimization*
Robert S. Cahn
- Frame Relay Applications: Business and Technology Case Studies*
James P. Cavanagh

For further information on these books and for a list of forthcoming titles, please visit our website at <http://www.mkp.com>

THIRD EDITION

Larry L. Peterson & Bruce S. Davie

COMPUTER NETWORKS

A Systems Approach



MORGAN KAUFMANN PUBLISHERS

AN IMPRINT OF ELSEVIER SCIENCE

AMSTERDAM BOSTON LONDON NEW YORK
OXFORD PARIS SAN DIEGO SAN FRANCISCO
SINGAPORE SYDNEY TOKYO

Senior Editor Rick Adams
Publishing Services Manager Simon Crump
Developmental Editor Karyn Johnson
Cover Design Ross Carron Design
Cover Image Vasco de Gama Bridge, Lisbon, Portugal
Composition/Illustration International Typesetting and Composition
Copyeditor Ken DellaPenta
Proofreader Jennifer McClain
Indexer Steve Rath
Printer Courier Corporation

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which Morgan Kaufmann Publishers is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Morgan Kaufmann Publishers
An Imprint of Elsevier Science
340 Pine Street, Sixth Floor
San Francisco, CA 94104-3205
www.mkp.com

© 2003 by Elsevier Science (USA)
All rights reserved
Printed in the United States of America

07 06 05 04 03 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, or otherwise—without the prior written permission of the publisher.

Library of Congress Control Number: xxxxxxxxxx
ISBN: 1-55860-832-X (Casebound)
ISBN: 1-55860-833-8 (Paperback)

This book is printed on acid-free paper.

To Lee Peterson and Robert Davie

This Page Intentionally Left Blank

FOREWORD

David Clark

Massachusetts Institute of Technology

This third edition represents another major upgrade to this classic networking book. The field continues to change fast, and new concepts emerge with amazing speed. This version expands its discussion of a lot of important new topics, including peer-to-peer networks, IPv6, overlay and content distribution networks, MPLS and switching, wireless and mobile technology, and more. It also contains an earlier and stronger focus on applications, which reflects the student and professional's increased familiarity with a wide range of networked applications. The book continues its tradition of giving you the facts you need to understand today's world.

But it has not lost track of its larger goal, to tell you not only the facts but the *why* behind the facts. The philosophy of the book remains the same: to be timely but timeless. What this book will teach you in today's networked world will give you the insight needed to work in tomorrow's landscape. And that is important, since there is no reason to believe that the evolution of networks is going to slow down anytime soon.

It is hard to remember what the world looked like only ten years ago. Back then the Internet was not really a commercial reality. Ten megabits per second was really fast. We didn't worry about spam and virus attacks—we left our computers unguarded and hardly worried. Those times were simpler, but today may be more exciting. And you better believe that tomorrow will be different from today: at least as exciting, with luck no less trustworthy, and certainly bigger, faster and filled with fresh innovation.

So I hope Larry and Bruce can relax for a little before they have to start the next revision. Meanwhile, use this book to learn about today and get ready for tomorrow. Have fun.

This Page Intentionally Left Blank

FOREWORD TO THE FIRST EDITION

David Clark

Massachusetts Institute of Technology

The term *spaghetti code* is universally understood as an insult. All good computer scientists worship the god of modularity, since modularity brings many benefits, including the all-powerful benefit of not having to understand all parts of a problem at the same time in order to solve it. Modularity thus plays a role in presenting ideas in a book, as well as in writing code. If a book's material is organized effectively—modularly—the reader can start at the beginning and actually make it to the end.

The field of network protocols is perhaps unique in that the “proper” modularity has been handed down to us in the form of an international standard: the seven-layer reference model of network protocols from the ISO. This model, which reflects a layered approach to modularity, is almost universally used as a starting point for discussions of protocol organization, whether the design in question conforms to the model or deviates from it.

It seems obvious to organize a networking book around this layered model. However, there is a peril to doing so, because the OSI model is not really successful at organizing the core concepts of networking. Such basic requirements as reliability, flow control, or security can be addressed at most, if not all, of the OSI layers. This fact has led to great confusion in trying to understand the reference model. At times it even requires a suspension of disbelief. Indeed, a book organized strictly according to a layered model has some of the attributes of spaghetti code.

Which brings us to this book. Peterson and Davie follow the traditional layered model, but they do not pretend that this model actually helps in the understanding of the big issues in networking. Instead, the authors organize discussion of fundamental concepts in a way that is independent of layering. Thus, after reading the book, readers will understand flow control, congestion control, reliability enhancement, data representation, and synchronization, and will separately understand the implications of addressing these issues in one or another of the traditional layers.

This is a timely book. It looks at the important protocols in use today—especially the Internet protocols. Peterson and Davie have a long involvement in and much experience with the Internet. Thus their book reflects not just the theoretical issues in

protocol design, but the real factors that matter in practice. The book looks at some of the protocols that are just emerging now, so the reader can be assured of an up-to-date perspective. But most importantly, the discussion of basic issues is presented in a way that derives from the fundamental nature of the problem, not the constraints of the layered reference model or the details of today's protocols. In this regard, what this book presents is both timely and timeless. The combination of real-world relevance, current examples, and careful explanation of fundamentals makes this book unique.

C O N T E N T S

Foreword	vii
Foreword to the First Edition	ix
Preface	xix
1 Foundation	
Problem: Building a Network	2
1.1 Applications	4
1.2 Requirements	6
1.2.1 Connectivity	7
1.2.2 Cost-Effective Resource Sharing	10
1.2.3 Support for Common Services	15
1.3 Network Architecture	19
1.3.1 Layering and Protocols	20
1.3.2 OSI Architecture	26
1.3.3 Internet Architecture	27
1.4 Implementing Network Software	30
1.4.1 Application Programming Interface (Sockets)	31
1.4.2 Example Application	33
1.4.3 Protocol Implementation Issues	36
1.5 Performance	40
1.5.1 Bandwidth and Latency	40
1.5.2 Delay × Bandwidth Product	44
1.5.3 High-Speed Networks	46
1.5.4 Application Performance Needs	48
1.6 Summary	50
Open Issue: Ubiquitous Networking	51
Further Reading	52
Exercises	55

2 Direct Link Networks

Problem: Physically Connecting Hosts	64
2.1 Hardware Building Blocks	66
2.1.1 Nodes	66
2.1.2 Links	67
2.2 Encoding (NRZ, NRZI, Manchester, 4B/5B)	75
2.3 Framing	79
2.3.1 Byte-Oriented Protocols (BISYNC, PPP, DDCMP)	80
2.3.2 Bit-Oriented Protocols (HDLC)	83
2.3.3 Clock-Based Framing (SONET)	84
2.4 Error Detection	88
2.4.1 Two-Dimensional Parity	89
2.4.2 Internet Checksum Algorithm	90
2.4.3 Cyclic Redundancy Check	92
2.5 Reliable Transmission	97
2.5.1 Stop-and-Wait	98
2.5.2 Sliding Window	101
2.5.3 Concurrent Logical Channels	110
2.6 Ethernet (802.3)	111
2.6.1 Physical Properties	112
2.6.2 Access Protocol	114
2.6.3 Experience with Ethernet	119
2.7 Token Rings (802.5, FDDI)	120
2.7.1 Physical Properties	121
2.7.2 Token Ring Media Access Control	122
2.7.3 Token Ring Maintenance	125
2.7.4 Frame Format	126
2.7.5 FDDI	127
2.8 Wireless (802.11)	131
2.8.1 Physical Properties	132
2.8.2 Collision Avoidance	133
2.8.3 Distribution System	134
2.8.4 Frame Format	136
2.9 Network Adaptors	137
2.9.1 Components	138
2.9.2 View from the Host	139
2.9.3 Memory Bottleneck	144
2.10 Summary	146

Open Issue: Does It Belong in Hardware?	147
Further Reading	148
Exercises	150

3 Packet Switching

Problem: Not All Networks Are Directly Connected	164
3.1 Switching and Forwarding	166
3.1.1 Datagrams	168
3.1.2 Virtual Circuit Switching	170
3.1.3 Source Routing	177
3.2 Bridges and LAN Switches	180
3.2.1 Learning Bridges	181
3.2.2 Spanning Tree Algorithm	185
3.2.3 Broadcast and Multicast	189
3.2.4 Limitations of Bridges	190
3.3 Cell Switching (ATM)	192
3.3.1 Cells	193
3.3.2 Segmentation and Reassembly	198
3.3.3 Virtual Paths	203
3.3.4 Physical Layers for ATM	204
3.3.5 ATM in the LAN	205
3.4 Implementation and Performance	210
3.4.1 Ports	212
3.4.2 Fabrics	216
3.5 Summary	220
Open Issue: The Future of ATM	221
Further Reading	221
Exercises	223

4 Internetworking

Problem: There Is More Than One Network	234
4.1 Simple Internetworking (IP)	236
4.1.1 What Is an Internetwork?	236
4.1.2 Service Model	238
4.1.3 Global Addresses	250
4.1.4 Datagram Forwarding in IP	252
4.1.5 Address Translation (ARP)	258
4.1.6 Host Configuration (DHCP)	263

4.1.7 Error Reporting (ICMP)	266
4.1.8 Virtual Networks and Tunnels	267
4.2 Routing	271
4.2.1 Network as a Graph	272
4.2.2 Distance Vector (RIP)	274
4.2.3 Link State (OSPF)	282
4.2.4 Metrics	291
4.2.5 Routing for Mobile Hosts	295
4.3 Global Internet	299
4.3.1 Subnetting	301
4.3.2 Classless Routing (CIDR)	306
4.3.3 Interdomain Routing (BGP)	308
4.3.4 Routing Areas	316
4.3.5 IP Version 6 (IPv6)	318
4.4 Multicast	331
4.4.1 Link-State Multicast	332
4.4.2 Distance-Vector Multicast	332
4.4.3 Protocol Independent Multicast (PIM)	336
4.5 Multiprotocol Label Switching (MPLS)	340
4.5.1 Destination-Based Forwarding	340
4.5.2 Explicit Routing	346
4.5.3 Virtual Private Networks and Tunnels	348
4.6 Summary	352
Open Issue: Deployment of IPV6	353
Further Reading	354
Exercises	355
 5 End-to-End Protocols	
Problem: Getting Processes to Communicate	374
5.1 Simple Demultiplexer (UDP)	376
5.2 Reliable Byte Stream (TCP)	378
5.2.1 End-to-End Issues	379
5.2.2 Segment Format	382
5.2.3 Connection Establishment and Termination	384
5.2.4 Sliding Window Revisited	389
5.2.5 Triggering Transmission	395
5.2.6 Adaptive Retransmission	397
5.2.7 Record Boundaries	401

5.2.8 TCP Extensions	402
5.2.9 Alternative Design Choices	403
5.3 Remote Procedure Call	405
5.3.1 Bulk Transfer (BLAST)	408
5.3.2 Request/Reply (CHAN)	414
5.3.3 Dispatcher (SELECT)	423
5.3.4 Putting It All Together (SunRPC, DCE)	424
5.4 Performance	431
5.5 Summary	434
Open Issue: Application-Specific Protocols	435
Further Reading	436
Exercises	437

6 Congestion Control and Resource Allocation

Problem: Allocating Resources	450
6.1 Issues in Resource Allocation	452
6.1.1 Network Model	452
6.1.2 Taxonomy	456
6.1.3 Evaluation Criteria	458
6.2 Queuing Disciplines	461
6.2.1 FIFO	462
6.2.2 Fair Queuing	463
6.3 TCP Congestion Control	468
6.3.1 Additive Increase/Multiplicative Decrease	468
6.3.2 Slow Start	471
6.3.3 Fast Retransmit and Fast Recovery	476
6.4 Congestion-Avoidance Mechanisms	478
6.4.1 DECbit	478
6.4.2 Random Early Detection (RED)	480
6.4.3 Source-Based Congestion Avoidance	486
6.5 Quality of Service	492
6.5.1 Application Requirements	493
6.5.2 Integrated Services (RSVP)	499
6.5.3 Differentiated Services (EF, AF)	509
6.5.4 ATM Quality of Service	513
6.5.5 Equation-Based Congestion Control	517
6.6 Summary	518

Open Issue: Inside versus Outside the Network	519
Further Reading	520
Exercises	521
 7 End-to-End Data	
Problem: What Do We Do with the Data?	534
7.1 Presentation Formatting	536
7.1.1 Taxonomy	537
7.1.2 Examples (XDR, ASN.1, NDR)	541
7.1.3 Markup Languages (XML)	545
7.2 Data Compression	548
7.2.1 Lossless Compression Algorithms	550
7.2.2 Image Compression (JPEG)	552
7.2.3 Video Compression (MPEG)	557
7.2.4 Transmitting MPEG over a Network	562
7.2.5 Audio Compression (MP3)	566
7.3 Summary	568
Open Issue: Computer Networks Meet Consumer Electronics	568
Further Reading	569
Exercises	570
 8 Network Security	
Problem: Securing the Data	578
8.1 Cryptographic Algorithms	580
8.1.1 Requirements	582
8.1.2 Secret Key Encryption (DES)	583
8.1.3 Public Key Encryption (RSA)	588
8.1.4 Message Digest Algorithms (MD5)	591
8.1.5 Implementation and Performance	593
8.2 Security Mechanisms	594
8.2.1 Authentication Protocols	594
8.2.2 Message Integrity Protocols	598
8.2.3 Public Key Distribution (X.509)	601
8.3 Example Systems	604
8.3.1 Pretty Good Privacy (PGP)	605
8.3.2 Secure Shell (SSH)	607
8.3.3 Transport Layer Security (TLS, SSL, HTTPS)	610
8.3.4 IP Security (IPSEC)	614

8.4	Firewalls	617
8.4.1	Filter-Based Firewalls	618
8.4.2	Proxy-Based Firewalls	620
8.4.3	Limitations	622
8.5	Summary	622
	Open Issue: Denial-of-Service Attacks	623
	Further Reading	624
	Exercises	625
9	Applications	
	Problem: Applications Need Their Own Protocols	632
9.1	Name Service (DNS)	634
9.1.1	Domain Hierarchy	635
9.1.2	Name Servers	636
9.1.3	Name Resolution	640
9.2	Traditional Applications	643
9.2.1	Electronic Mail (SMTP, MIME, IMAP)	644
9.2.2	World Wide Web (HTTP)	650
9.2.3	Network Management (SNMP)	657
9.3	Multimedia Applications	660
9.3.1	Real-time Transport Protocol (RTP)	660
9.3.2	Session Control and Call Control (SDP, SIP, H.323)	671
9.4	Overlay Networks	680
9.4.1	Routing Overlays	682
9.4.2	Peer-to-Peer Networks	690
9.4.3	Content Distribution Networks	698
9.5	Summary	704
	Open Issue: New Network Architecture	704
	Further Reading	705
	Exercises	707
	Glossary	715
	Bibliography	743
	Solutions to Selected Exercises	763
	Index	777
	About the Authors	810

just a program running on the same machine as the user's mailbox resides, in which case it simply reads and writes the file that implements the mailbox. In other cases, the user accesses his or her mailbox from a remote machine using yet another protocol, such as the Post Office Protocol (POP) or the Internet Message Access Protocol (IMAP). It is beyond the scope of this book to discuss the user interface aspects of the mail reader, but it is definitely within our scope to talk about the access protocol. We consider IMAP, in particular.

IMAP is similar to SMTP in many ways. It is a client/server protocol running over TCP, where the client (running on the user's desktop machine) issues commands in the form of <CRLF>-terminated ASCII text lines and the mail server (running on the machine that maintains the user's mailbox) responds in kind. The exchange begins with the client authenticating him- or herself, and identifying the mailbox he or she wants to access. This can be represented by the simple state transition diagram shown in Figure 9.7. In this diagram, LOGIN, AUTHENTICATE, SELECT, EXAMINE, CLOSE, and LOGOUT are example commands that the client can issue, while OK is one possible server response. Other common commands include FETCH, STORE, DELETE, and EXPUNGE, with the obvious meanings. Additional server responses include NO (client does not have permission to perform that operation) and BAD (command is ill formed).

When the user asks to FETCH a message, the server returns it in MIME format and the mail reader decodes it. In addition to the message itself, IMAP also defines a set of message *attributes* that are exchanged as part of other commands, independent of transferring the message itself. Message attributes include information like the size of the message, but more interestingly, various *flags* associated with the message (e.g., Seen, Answered, Deleted, and Recent). These flags are used to keep the client and server synchronized; that is, when the user deletes a message in the mail reader, the client needs to report this fact to the mail server. Later, should the user decide to expunge all deleted messages, the client issues an EXPUNGE command to the server, which knows to actually remove all earlier deleted messages from the mailbox.

Finally, note that when the user replies to a message, or sends a new message, the mail reader does not forward the message from the client's desktop machine to the mail server using IMAP, but it instead uses SMTP. This means that the user's mail server is effectively the first mail gateway traversed along the path from the desktop to the recipient's mailbox.

9.2.2 World Wide Web (HTTP)

The World Wide Web has been so successful and has made the Internet accessible to so many people that sometimes it seems to be synonymous with the Internet. One helpful way to think of the Web is as a set of cooperating clients and servers, all of whom speak the same language: HTTP. Most people are exposed to the Web through

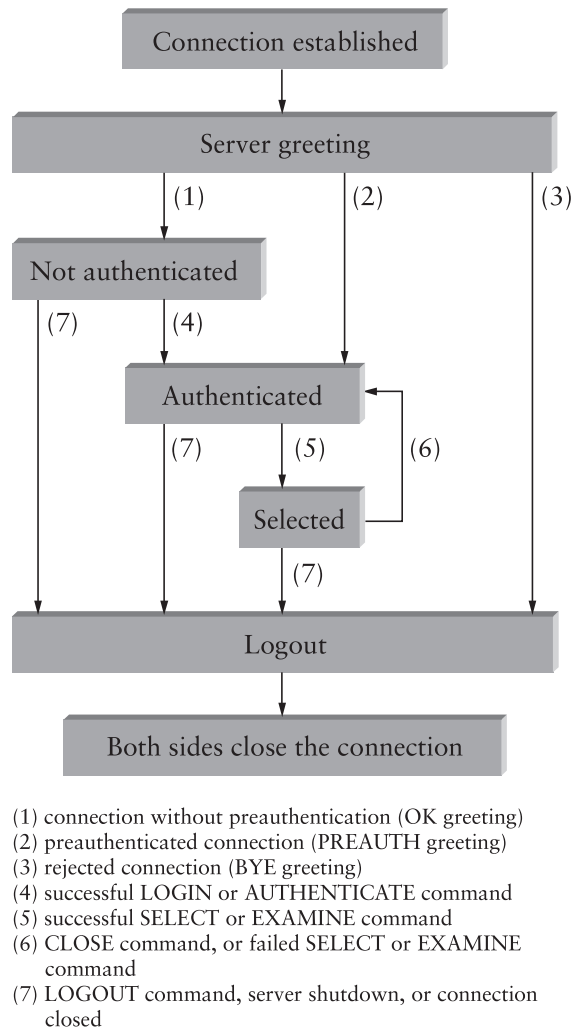


Figure 9.7 IMAP state transition diagram.

a graphical client program, or Web browser, like Netscape or Explorer. Figure 9.8 shows the Netscape browser in use, displaying a page of information from a project called PlanetLab.

Any Web browser has a function that allows the user to “open a URL.” URLs (uniform resource locators) provide information about the location of objects on the



Figure 9.8 The Netscape Web browser.

Web; they look like the following:

<http://www.cs.princeton.edu/index.html>

If you opened that particular URL, your Web browser would open a TCP connection to the Web server at a machine called `www.cs.princeton.edu` and immediately retrieve

and display the file called `index.html`. Most files on the Web contain images and text, and some have audio and video clips. They also include URLs that point to other files, and your Web browser will have some way in which you can recognize URLs and ask the browser to open them. These embedded URLs are called *hypertext links*. When you ask your Web browser to open one of these embedded URLs (e.g., by pointing and clicking on it with a mouse), it will open a new connection and retrieve and display a new file. This is called “following a link.” It thus becomes very easy to hop from one machine to another around the network, following links to all sorts of information.

When you select to view a page, your browser (the client) fetches the page from the server using HTTP running over TCP. Like SMTP, HTTP is a text-oriented protocol. At its core, each HTTP message has the general form

```
START_LINE <CRLF>
MESSAGE_HEADER <CRLF>
<CRLF>
MESSAGE_BODY <CRLF>
```

where as before, `<CRLF>` stands for carriage-return-line-feed. The first line (`START_LINE`) indicates whether this is a request message or a response message. In effect, it identifies the “remote procedure” to be executed (in the case of a request message) or the “status” of the request (in the case of a response message). The next set of lines specifies a collection of options and parameters that qualify the request or response. There are zero or more of these `MESSAGE_HEADER` lines—the set is terminated by a blank line—each of which looks like a header line in an email message. HTTP defines many possible header types, some of which pertain to request messages, some to response messages, and some to the data carried in the message body. Instead of giving the full set of possible header types, though, we just give a handful of representative examples. Finally, after the blank line comes the contents of the requested message (`MESSAGE_BODY`); this part of the message is typically empty for request messages.

Request Messages

The first line of an HTTP request message specifies three things: the operation to be performed, the Web page the operation should be performed on, and the version of HTTP being used. Although HTTP defines a wide assortment of possible request operations—including “write” operations that allow a Web page to be posted on a server—the two most common operations are `GET` (fetch the specified Web page) and `HEAD` (fetch status information about the specified Web page). The former is obviously used when your browser wants to retrieve and display a Web page. The latter is used to test the validity of a hypertext link or to see if a particular page has been

Operation	Description
OPTIONS	request information about available options
GET	retrieve document identified in URL
HEAD	retrieve metainformation about document identified in URL
POST	give information (e.g., annotation) to server
PUT	store document under specified URL
DELETE	delete specified URL
TRACE	loopback request message
CONNECT	for use by proxies

Table 9.1 HTTP request operations.

modified since the browser last fetched it. The full set of operations is summarized in Table 9.1.

For example, the **START_LINE**

```
GET http://www.cs.princeton.edu/index.html HTTP/1.1
```

says that the client wants the server on host `www.cs.princeton.edu` to return the page named `index.html`. This particular example uses an *absolute* URL. It is also possible to use a *relative* identifier and specify the host name in one of the **MESSAGE_HEADER** lines; for example,

```
GET index.html HTTP/1.1
Host: www.cs.princeton.edu
```

Here, **Host** is one of the possible **MESSAGE_HEADER** fields. One of the more interesting of these is **If-Modified-Since**, which gives the client a way to conditionally request a Web page—the server returns the page only if it has been modified since the time specified in that header line.

Response Messages

Like request messages, response messages begin with a single **START_LINE**. In this case, the line specifies the version of HTTP being used, a three-digit code indicating whether or not the request was successful, and a text string giving the reason for the response. For example, the **START_LINE**

```
HTTP/1.1 202 Accepted
```

Code	Type	Example Reasons
1xx	Informational	request received, continuing process
2xx	Success	action successfully received, understood, and accepted
3xx	Redirection	further action must be taken to complete the request
4xx	Client Error	request contains bad syntax or cannot be fulfilled
5xx	Server Error	server failed to fulfill an apparently valid request

Table 9.2 Five types of HTTP result codes.

indicates that the server was able to satisfy the request, while

```
HTTP/1.1 404 Not Found
```

indicates that it was not able to satisfy the request because the page was not found. There are five general types of response codes, with the first digit of the code indicating its type. Table 9.2 summarizes the five types of codes.

Also similar to request messages, response messages can contain one or more **MESSAGE_HEADER** lines. These lines relay additional information back to the client. For example, the **Location** header line specifies that the requested URL is available at another location. Thus, if the Princeton CS Department Web page had moved from <http://www.cs.princeton.edu/index.html> to <http://www.princeton.edu/cs/index.html>, for example, then the server at the original address might respond with

```
HTTP/1.1 301 Moved Permanently
Location: http://www.princeton.edu/cs/index.html
```

In the common case, the response message will also carry the requested page. This page is an HTML document, but since it may carry nontextual data (e.g., a GIF image), it is encoded using MIME (see Section 9.2.1). Certain of the **MESSAGE_HEADER** lines give attributes of the page contents, including **Content-Length** (number of bytes in the contents), **Expires** (time at which the contents are considered stale), and **Last-Modified** (time at which the contents were last modified at the server).

TCP Connections

The original version of HTTP (1.0) established a separate TCP connection for each data item retrieved from the server. It's not too hard to see how this was a very inefficient mechanism: Connection setup and teardown messages had to be exchanged between

the client and server even if all the client wanted to do was verify that it had the most recent copy of a page. Thus, retrieving a page that included some text and a dozen icons or other small graphics would result in 13 separate TCP connections being established and closed.

The most important improvement in the latest version of HTTP (1.1) is to allow *persistent connections*—the client and server can exchange multiple request/response messages over the same TCP connection. Persistent connections have two advantages. First, they obviously eliminate the connection setup overhead, thereby reducing the load on the server, the load on the network caused by the additional TCP packets, and the delay perceived by the user. Second, because a client can send multiple request messages down a single TCP connection, TCP's congestion window mechanism is able to operate more efficiently. This is because it's not necessary to go through the slow start phase for each page.

Persistent connections do not come without a price, however. The problem is that neither the client nor server necessarily knows how long to keep a particular TCP connection open. This is especially critical on the server, which might be asked to keep connections open on behalf of thousands of clients. The solution is that the server must time out and close a connection if it has received no requests on the connection for a period of time. Also, both the client and server must watch to see if the other side has elected to close the connection, and they must use that information as a signal that they should close their side of the connection as well. (Recall that both sides must close a TCP connection before it is fully terminated.)

Caching

One of the most active areas of research (and entrepreneurship) in the Internet today is how to effectively cache Web pages. Caching has many benefits. From the client's perspective, a page that can be retrieved from a nearby cache can be displayed much more quickly than if it has to be fetched from across the world. From the server's perspective, having a cache intercept and satisfy a request reduces the load on the server.

Caching can be implemented in many different places. For example, a user's browser can cache recently accessed pages, and simply display the cached copy if the user visits the same page again. As another example, a site can support a single sitewide cache. This allows users to take advantage of pages previously downloaded by other users. Closer to the middle of the Internet, ISPs can cache pages. Note that in the second case, the users within the site most likely know what machine is caching pages on behalf of the site, and they configure their browsers to connect directly to the caching host. This node is sometimes called a *proxy*. In contrast, the sites that connect to the ISP are probably not aware that the ISP is caching pages. It simply happens to be

the case that HTTP requests coming out of the various sites pass through a common ISP router. This router can peek inside the request message and look at the URL for the requested page. If it has the page in its cache, it returns it. If not, it forwards the request to the server and watches for the response to fly by in the other direction. When it does, the router saves a copy in the hope that it can use it to satisfy a future request.

No matter where pages are cached, the ability to cache Web pages is important enough that HTTP has been designed to make the job easier. The trick is that the cache needs to make sure it is not responding with an out-of-date version of the page. For example, the server assigns an expiration date (the Expires header field) to each page it sends back to the client (or to a cache between the server and client). The cache remembers this date and knows that it need not reverify the page each time it is requested until after that expiration date has passed. After that time (or if that header field is not set) the cache can use the HEAD or conditional GET operation (GET with If-Modified-Since header line) to verify that it has the most recent copy of the page. More generally, there is a set of “cache directives” that must be obeyed by all caching mechanisms along the request/response chain. These directives specify whether or not a document can be cached, how long it can be cached, how fresh a document must be, and so on.

9.2.3 Network Management (SNMP)

A network is a complex system, both in terms of the number of nodes that are involved and in terms of the suite of protocols that can be running on any one node. Even if you restrict yourself to worrying about the nodes within a single administrative domain, such as a campus, there might be dozens of routers and hundreds—or even thousands—of hosts to keep track of. If you think about all the state that is maintained and manipulated on any one of those nodes—for example, address translation tables, routing tables, TCP connection state, and so on—then it is easy to become depressed about the prospect of having to manage all of this information.

It is easy to imagine wanting to know about the state of various protocols on different nodes. For example, you might want to monitor the number of IP datagram reassemblies that have been aborted, so as to determine if the timeout that garbage collects partially assembled datagrams needs to be adjusted. As another example, you might want to keep track of the load on various nodes (i.e., the number of packets sent or received) so as to determine if new routers or links need to be added to the network. Of course, you also have to be on the watch for evidence of faulty hardware and misbehaving software.

What we have just described is the problem of network management, an issue that pervades the entire network architecture. Since the nodes we want to keep track of are distributed, our only real option is to use the network to manage the network.