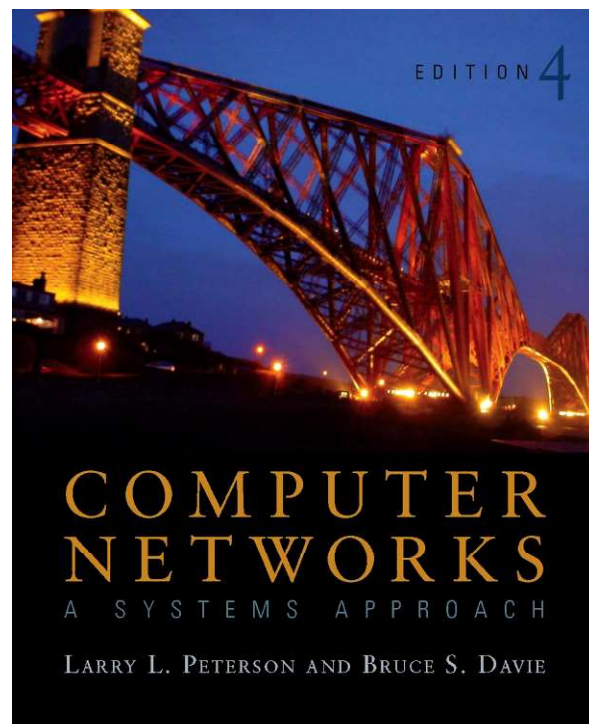




EDITION 4

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, 4e

Larry L. Peterson and Bruce S. Davie

Network Routing: Algorithms, Protocols, and Architectures

Deepankar Medhi and Karthikeyan Ramaswami

Deploying IP and MPLS QoS for Multiservice Networks: Theory and Practice

John Evans and Clarence Filsfil

Traffic Engineering and QoS Optimization of Integrated Voice & Data Networks

Gerald R. Ash

IPv6 Core Protocols Implementation

Qing Li, Tatuya Jinmei, and Keiichi Shima

Smart Phone and Next-Generation Mobile Computing

Pei Zheng and Lionel Ni

GMPLS: Architecture and Applications

Adrian Farrel and Igor Bryskin

Network Security: A Practical Approach

Jan L. Harrington

Content Networking: Architecture, Protocols, and Practice

Markus Hofmann and Leland R. Beaumont

Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices

George Varghese

Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS

Jean Philippe Vasseur, Mario Pickavet, and Piet

Demeester

Routing, Flow, and Capacity Design in Communication and Computer Networks

Michał Pióro and Deepankar Medhi

Wireless Sensor Networks: An Information Processing Approach

Feng Zhao and Leonidas Guibas

Communication Networking: An Analytical Approach

Anurag Kumar, D. Manjunath, and Joy Kuri

The Internet and Its Protocols: A Comparative Approach

Adrian Farrel

Modern Cable Television Technology: Video, Voice, and Data Communications, 2e

Walter Ciciora, James Farmer, David Large, and

Michael Adams

Bluetooth Application Programming with the Java APIs

C Bala Kumar, Paul J. Kline, and Timothy J.

Thompson

Policy-Based Network Management: Solutions for the Next Generation

John Strassner

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

Wide Area Network Design: Concepts and Tools for Optimization

Robert S. Cahn

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

EDITION 4

COMPUTER NETWORKS

A S Y S T E M S A P P R O A C H

LARRY L. PETERSON & BRUCE S. DAVIE



AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO
Morgan Kaufmann Publishers is an imprint of Elsevier



MORGAN KAUFMANN PUBLISHERS

Acquisitions Editor
Publishing Services Manager
Production Editor
Associate Acquisitions Editor
Design Direction
Composition
Copyeditor
Proofreader
Indexer
Interior printer
Cover printer

Rick Adams
George Morrison
Dawnmarie Simpson
Rachel Roumeliotis
Louis Forgione
VTEX
Multiscience Press, Inc.
Jodie Allen
Multiscience Press, Inc.
Courier Westford
Phoenix Color, Inc.

Morgan Kaufmann Publishers is an imprint of Elsevier.
500 Sansome Street, Suite 400, San Francisco, CA 94111

This book is printed on acid-free paper.

© 2007, Elsevier, Inc. All rights reserved.

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.

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, scanning, or otherwise—without prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, E-mail: permissions@elsevier.com. You may also complete your request online via the Elsevier homepage (<http://elsevier.com>), by selecting "Support & Contact" then "Copyright and Permission" and then "Obtaining Permissions."

ISBN 13: 978-0-12-370548-8 (Case bound)

ISBN 10: 0-12-370548-7 (Case bound)

ISBN 13: 978-0-12-374013-7 (Paperback)

ISBN 10: 0-12-374013-4 (Paperback)

Library of Congress Cataloging-in-Publication Data

Peterson, Larry L.

Computer networks : a systems approach / Larry L. Peterson & Bruce S. Davie – 4th ed.

p. cm.

Includes bibliographical references and index.

ISBN-13: 978-0-12-370548-8 (hardcover : alk. paper)

ISBN-10: 0-12-370548-7 (hardcover : alk. paper)

ISBN-13: 978-0-12-374013-7 (pbk. : alk. paper)

ISBN-10: 0-12-374013-4 (pbk. : alk. paper) 1. Computer networks. I.

Davie, Bruce S. II. Title.

TK5105.5.P479 2007

004.6'5–dc22

2006102454

For information on all Morgan Kaufmann publications, visit our
Web site at www.mkp.com or www.books.elsevier.com

Printed in the United States of America.
06 07 08 09 10 5 4 3 2 1

Working together to grow
libraries in developing countries

www.elsevier.com | www.bookaid.org | www.sabre.org

ELSEVIER **BOOK AID** **Sabre Foundation**
International

To Lee Peterson and Robert Davie

CONTENTS

Foreword	vii
Foreword to the First Edition	ix
Preface	xi
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	11
1.2.3 Support for Common Services	14
1.3 Network Architecture	19
1.3.1 Layering and Protocols	20
1.3.2 OSI Architecture	26
1.3.3 Internet Architecture	28
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	37
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	71
2.2 Encoding (NRZ, NRZI, Manchester, 4B/5B)	79
2.3 Framing	84
2.3.1 Byte-Oriented Protocols (PPP)	84
2.3.2 Bit-Oriented Protocols (HDLC)	87
2.3.3 Clock-Based Framing (SONET)	89
2.4 Error Detection	92
2.4.1 Two-Dimensional Parity	93
2.4.2 Internet Checksum Algorithm	94
2.4.3 Cyclic Redundancy Check	96
2.5 Reliable Transmission	101
2.5.1 Stop-and-Wait	102
2.5.2 Sliding Window	105
2.5.3 Concurrent Logical Channels	115
2.6 Ethernet (802.3)	116
2.6.1 Physical Properties	116
2.6.2 Access Protocol	119
2.6.3 Experience with Ethernet	123
2.7 Rings (802.5, FDDI, RPR)	124
2.7.1 Token Ring Media Access Control	127
2.7.2 Token Ring Maintenance	129
2.7.3 FDDI	130
2.7.4 Resilient Packet Ring (802.17)	131
2.8 Wireless	133
2.8.1 Bluetooth (802.15.1)	136
2.8.2 Wi-Fi (802.11)	137
2.8.3 WiMAX (802.16)	143
2.8.4 Cell Phone Technologies	145
2.9 Summary	147
Open Issue: Sensor Networks	148
Further Reading	149
Exercises	151

3 Packet Switching

Problem: Not All Networks Are Directly Connected	166
3.1 Switching and Forwarding	168

3.1.1	Datagrams	170
3.1.2	Virtual Circuit Switching	172
3.1.3	Source Routing	179
3.2	Bridges and LAN Switches	183
3.2.1	Learning Bridges	184
3.2.2	Spanning Tree Algorithm	187
3.2.3	Broadcast and Multicast	192
3.2.4	Limitations of Bridges	193
3.3	Cell Switching (ATM)	195
3.3.1	Cells	195
3.3.2	Segmentation and Reassembly	200
3.3.3	Virtual Paths	205
3.3.4	Physical Layers for ATM	206
3.4	Implementation and Performance	208
3.4.1	Ports	210
3.4.2	Fabrics	214
3.5	Summary	218
	Open Issue: The Future of Switching	219
	Further Reading	219
	Exercises	221
4	Internetworking	
	Problem: There Is More Than One Network	232
4.1	Simple Internetworking (IP)	234
4.1.1	What Is an Internetwork?	234
4.1.2	Service Model	236
4.1.3	Global Addresses	248
4.1.4	Datagram Forwarding in IP	250
4.1.5	Address Translation (ARP)	254
4.1.6	Host Configuration (DHCP)	259
4.1.7	Error Reporting (ICMP)	262
4.1.8	Virtual Networks and Tunnels	262
4.2	Routing	266
4.2.1	Network as a Graph	268
4.2.2	Distance Vector (RIP)	269
4.2.3	Link State (OSPF)	277
4.2.4	Metrics	286
4.2.5	Routing for Mobile Hosts	289
4.2.6	Router Implementation	294
4.3	Global Internet	297

4.3.1 Subnetting	299
4.3.2 Classless Routing (CIDR)	303
4.3.3 Interdomain Routing (BGP)	306
4.3.4 Routing Areas	316
4.3.5 IP Version 6 (IPv6)	318
4.4 Multicast	329
4.4.1 Multicast Addresses	331
4.4.2 Multicast Routing (DVMRP, PIM, MSDP)	332
4.5 Multiprotocol Label Switching	343
4.5.1 Destination-Based Forwarding	344
4.5.2 Explicit Routing	350
4.5.3 Virtual Private Networks and Tunnels	352
4.6 Summary	356
Open Issue: Deployment of IPv6	358
Further Reading	359
Exercises	360
5 End-to-End Protocols	
Problem: Getting Processes to Communicate	380
5.1 Simple Demultiplexer (UDP)	382
5.2 Reliable Byte Stream (TCP)	384
5.2.1 End-to-End Issues	385
5.2.2 Segment Format	387
5.2.3 Connection Establishment and Termination	390
5.2.4 Sliding Window Revisited	394
5.2.5 Triggering Transmission	400
5.2.6 Adaptive Retransmission	403
5.2.7 Record Boundaries	407
5.2.8 TCP Extensions	408
5.2.9 Alternative Design Choices	410
5.3 Remote Procedure Call	411
5.3.1 RPC Fundamentals	412
5.3.2 RPC Implementations (SunRPC, DCE)	419
5.4 Transport for Real-Time Applications (RTP)	426
5.4.1 Requirements	428
5.4.2 RTP Details	429
5.4.3 Control Protocol	433
5.5 Performance	437
5.6 Summary	440
Open Issue: Application-Specific Protocols	441

Further Reading	442
Exercises	443

6 Congestion Control and Resource Allocation

Problem: Allocating Resources	456
6.1 Issues in Resource Allocation	458
6.1.1 Network Model	458
6.1.2 Taxonomy	462
6.1.3 Evaluation Criteria	464
6.2 Queuing Disciplines	467
6.2.1 FIFO	468
6.2.2 Fair Queuing	469
6.3 TCP Congestion Control	474
6.3.1 Additive Increase/Multiplicative Decrease	474
6.3.2 Slow Start	477
6.3.3 Fast Retransmit and Fast Recovery	483
6.4 Congestion-Avoidance Mechanisms	486
6.4.1 DECbit	486
6.4.2 Random Early Detection (RED)	487
6.4.3 Source-Based Congestion Avoidance	493
6.5 Quality of Service	499
6.5.1 Application Requirements	500
6.5.2 Integrated Services (RSVP)	506
6.5.3 Differentiated Services (EF, AF)	516
6.5.4 Equation-Based Congestion Control	522
6.6 Summary	524
Open Issue: Inside versus Outside the Network	525
Further Reading	526
Exercises	527

7 End-to-End Data

Problem: What Do We Do with the Data?	542
7.1 Presentation Formatting	544
7.1.1 Taxonomy	545
7.1.2 Examples (XDR, ASN.1, NDR)	549
7.1.3 Markup Languages (XML)	553
7.2 Data Compression	557
7.2.1 Lossless Compression Algorithms	559
7.2.2 Image Compression (JPEG)	561
7.2.3 Video Compression (MPEG)	566
7.2.4 Transmitting MPEG over a Network	571

7.2.5 Audio Compression (MP3)	575
7.3 Summary	576
Open Issue: Computer Networks Meet Consumer Electronics	577
Further Reading	578
Exercises	579
8 Network Security	
Problem: Security Attacks	586
8.1 Cryptographic Tools	589
8.1.1 Principles of Ciphers	589
8.1.2 Symmetric-Key Ciphers	591
8.1.3 Public-Key Ciphers	593
8.1.4 Authenticators	595
8.2 Key Predistribution	599
8.2.1 Predistribution of Public Keys	599
8.2.2 Predistribution of Symmetric Keys	604
8.3 Authentication Protocols	604
8.3.1 Originality and Timeliness Techniques	605
8.3.2 Public-Key Authentication Protocols	606
8.3.3 Symmetric-Key Authentication Protocols	607
8.3.4 Diffie-Hellman Key Agreement	611
8.4 Secure Systems	613
8.4.1 Pretty Good Privacy (PGP)	613
8.4.2 Secure Shell (SSH)	615
8.4.3 Transport Layer Security (TLS, SSL, HTTPS)	618
8.4.4 IP Security (IPsec)	622
8.4.5 Wireless Security (802.11i)	625
8.5 Firewalls	626
8.5.1 Strengths and Weaknesses of Firewalls	629
8.6 Summary	631
Open Issue: Denial-of-Service Attacks	632
Further Reading	633
Exercises	634
9 Applications	
Problem: Applications Need Their Own Protocols	640
9.1 Traditional Applications	642
9.1.1 Electronic Mail (SMTP, MIME, IMAP)	643
9.1.2 World Wide Web (HTTP)	650
9.1.3 Name Service (DNS)	657
9.1.4 Network Management (SNMP)	666

9.2 Web Services	668
9.2.1 Custom Application Protocols (WSDL, SOAP)	670
9.2.2 A Generic Application Protocol (REST)	676
9.3 Multimedia Applications	678
9.3.1 Session Control and Call Control (SDP, SIP, H.323)	679
9.3.2 Resource Allocation for Multimedia Applications	688
9.4 Overlay Networks	693
9.4.1 Routing Overlays	695
9.4.2 Peer-to-Peer Networks (Gnutella, BitTorrent)	702
9.4.3 Content Distribution Networks	714
9.5 Summary	719
Open Issue: New Network Architecture	720
Further Reading	721
Exercises	722
 Solutions to Select Exercises	 729
Glossary	743
Bibliography	769
Index	785

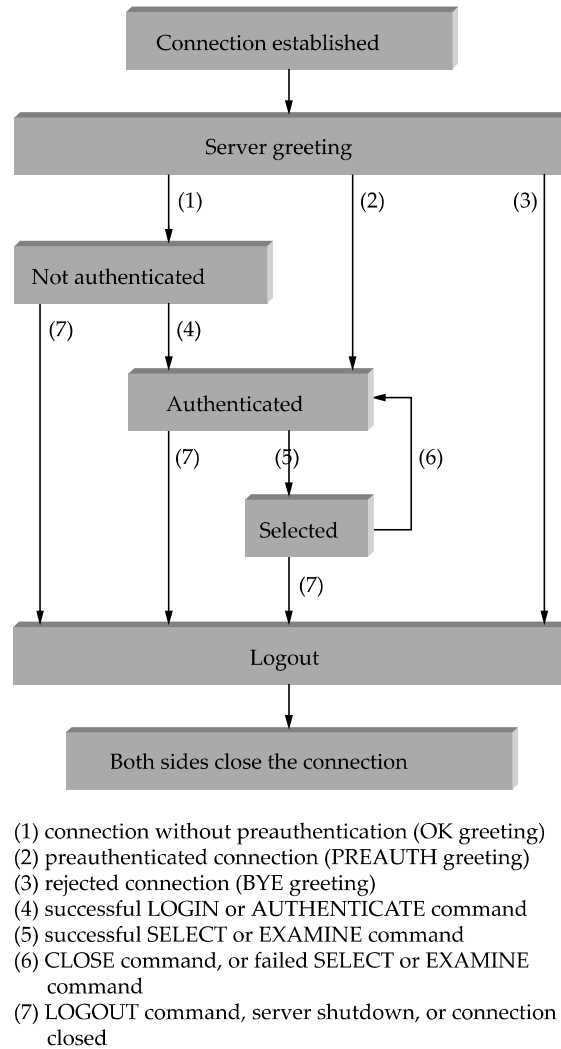


Figure 9.2 IMAP state transition diagram.

9.1.2 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 a graphical

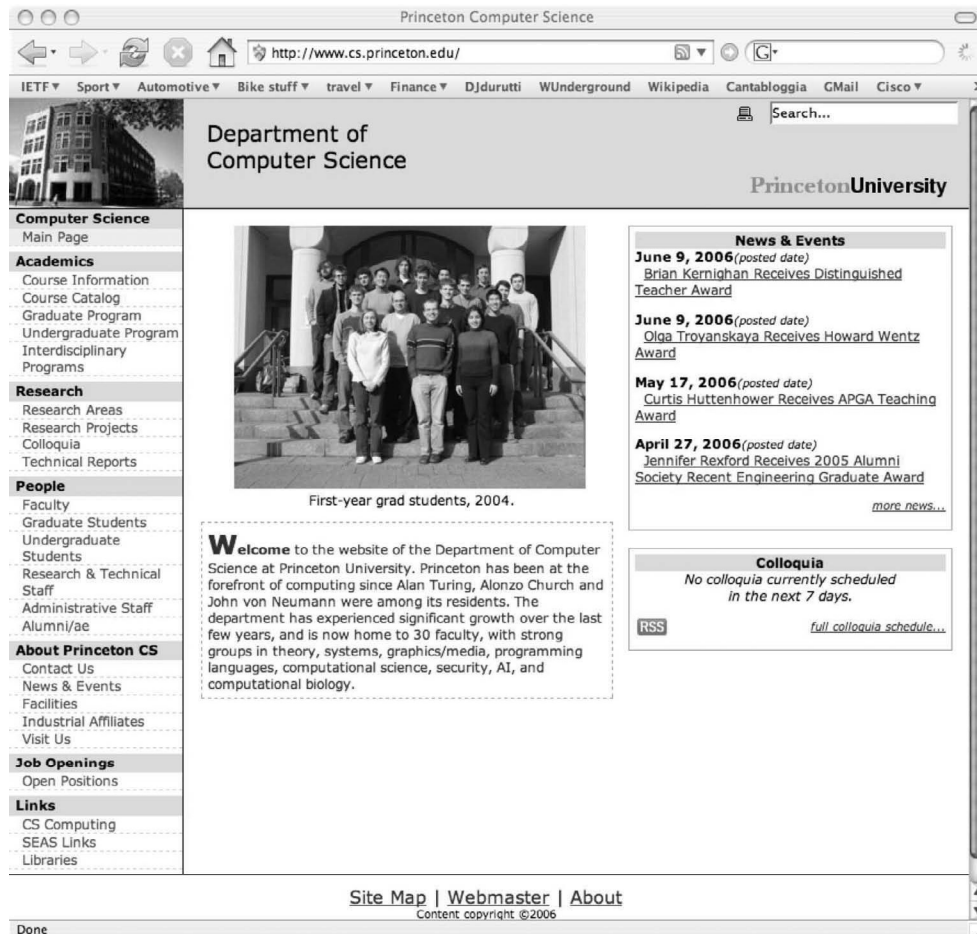


Figure 9.3 The Firefox web browser.

client program, or web browser, like Netscape, Firefox, or Explorer. Figure 9.3 shows the Firefox browser in use, displaying a page of information from Princeton University.

Any web browser has a function that allows the user to open a URL. URLs provide information about the location of objects on the 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 other objects such as 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 specify 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 modified since the browser last fetched it. The full set of operations is summarized in Table 9.1.

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.

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
```

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

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.

```
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 9.1.1). Certain **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).

Uniform Resource Identifiers

The URLs that HTTP uses as addresses are one type of *uniform resource identifier (URI)*. A URI is a character string that identifies a resource, where a resource can be anything that has identity, such as a document, an image, or a service.

The format of URIs allows various more-specialized kinds of resource identifiers to be incorporated into the URI space of identifiers. The first part of a URI is a *scheme* that names a particular way of identifying a certain kind of resource, such as **mailto** for

email addresses or **file** for file names. The second part of a URI, separated from the first part by a colon, is the *scheme-specific part*. It is a resource identifier consistent with the scheme in the first part, as in the URIs

```
mailto:santa@northpole.org
```

and

```
file:///C:/foo.html
```

A resource doesn't have to be retrievable or accessible. Even human beings and corporations can be resources. A more concrete example is the **mid** scheme for message IDs. Hence, URIs are not always some kind of address for locating the resource; they can be purely identifiers. Furthermore, a URI qualifies as a URL only if it is *intended* to be used to locate the resource. Even if a particular URI appears to be an address, such as a URI that uses the **http** scheme, the URI is not considered a URL unless it is intended to be used to locate the resource. For example, XML namespaces (Section 7.1.3) are identified by URIs that use the **http** scheme but are nonetheless not URLs since there is no requirement that the URI give the location of any resource related to the namespace.

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 opened 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 site-wide 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 are 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.