Apache

The Definitive Guide

Apache

The Definitive Guide

Second Edition

Ben Laurie and Peter Laurie

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

Apache: The Definitive Guide, Second Edition
by Ben Laurie and Peter Laurie

Printed in the United States of America.

Published by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

Editor: Robert Denn

Production Editor: Madeleine Newell

Printing History:

March 1997:	First Edition.
February 1999:	Second Edition.

ISBN: 1-56592-528-9 [12/99]
[M]

Table of Contents

Preface

Apache: The Definitive Guide is principally about the Apache web server software. We explain what a web server is and how it works, but our assumption is that most of our readers have used the World Wide Web and understand in practical terms how it works, and that they are now thinking about running their own servers to offer material to the hungry masses.

This book takes the reader through the process of acquiring, compiling, installing, configuring, and modifying Apache. We exercise most of the package's functions by showing a set of example sites that take a reasonably typical web business—in our case, a postcard publisher—through a process of development and increasing complexity. However, we have deliberately not tried to make each site more complicated than the last. Most of the chapters refer to an illustrative site that is as simple as we could make it. Each site is pretty well self-contained so that the reader can refer to it while following the text without having to disentangle the meat there from extraneous vegetables. If desired, it is perfectly possible to install and run each site on a suitable system.

Perhaps it is worth saying what this book is *not*. It is not a manual, in the sense of formally documenting every command—such a manual exists on the Apache site and has been much improved with Version 1.3; we assume that if you want to use Apache, you will download it and keep it at hand. Rather, if the manual is a roadmap that tells you how to get somewhere, this book tries to be a tourist guide that tells you why you might want to make the journey.

It also is *not* a book about HTML or creating web pages, or one about web security or even about running a web site. These are all complex subjects that should either be treated thoroughly or left alone. A compact, readable book that dealt *thoroughly* with all these topics would be most desirable.

A webmaster's library, however, is likely to be much bigger. It might include books on the following topics:

- The Web and how it works
- HTML—what you can do with it
- How to decide what sort of web site you want, how to organize it, and how to protect it
- How to implement the site you want using one of the available servers (for instance, Apache)
- Handbooks on Java, Perl, and other languages
- Security

Apache: The Definitive Guide is just one of the six or so possible titles in the fourth category.

Apache is a versatile package and is becoming more versatile every day, so we have not tried to illustrate every possible combination of commands; that would require a book of a million pages or so. Rather, we have tried to suggest lines of development that a typical webmaster should be able to follow once an understanding of the basic concepts is achieved.

As with the first edition, writing the book was something of a race with Apache's developers. We wanted to be ready as soon as Version 1.3 was stable, but not before the developers had finished adding new features. Unfortunately, although 1.3 was in "feature freeze" from early 1998 on, we could not be sure that new features might not become necessary to fix newly discovered problems.

In many of the examples that follow, the motivation for what we make Apache do is simple enough and requires little explanation (for example, the different index formats in Chapter 7). Elsewhere, we feel that the webmaster needs to be aware of wider issues (for instance, the security issues discussed in Chapter 13) before making sensible decisions about his or her site's configuration, and we have not hesitated to branch out to deal with them.

Who Wrote Apache, and Why?

Apache gets its name from the fact that it consists of some existing code plus some *patches*. The FAQ* thinks that this is cute; others may think it's the sort of joke that

* FAQ is netspeak for Frequently Asked Questions. Most sites/subjects have an FAQ file that tells you what the thing is, why it is, and where it is going. It is perfectly reasonable for the newcomer to ask for the FAQ to look up anything new to him or her, and indeed this is a sensible thing to do, since it reduces the number of questions asked. Apache's FAQ can be found at *http://www.apache.org/docs/FAQ.html*.

gets programmers a bad name. A more responsible group thinks that Apache is an appropriate title because of the resourcefulness and adaptability of the American Indian tribe.

You have to understand that Apache is free to its users and is written by a team of volunteers who do not get paid for their work. Whether or not they decide to incorporate your or anyone else's ideas is entirely up to them. If you don't like this, feel free to collect a team and write your own web server.

The first web server was built by the British physicist Tim Berners-Lee at CERN, the European Centre for Nuclear Research at Geneva, Switzerland. The immediate ancestor of Apache was built by the U.S. government in the person of NCSA, the National Center for Supercomputing Applications. This fine body is not to be confused with the National Computing Security Agency or the North Carolina Schools Association. Because this code was written with (American) taxpayers' money, it is available to all; you can, if you like, download the source code in C from *www.ncsa.uiuc.edu*, paying due attention to the license conditions.

There were those who thought that things could be done better, and in the FAQ for Apache (at *http://www.apache.org*) we read:

> ...Apache was originally based on code and ideas found in the most popular HTTP server of the time, NCSA httpd 1.3 (early 1995).

That phrase "of the time" is nice. It usually refers to good times back in the 1700s or the early days of technology in the 1900s. But here it means back in the deliquescent bogs of a few years ago!

While the Apache site is open to all, Apache is written by an invited group of (we hope) reasonably good programmers. One of the authors of this book, Ben, is a member of this group.

Why do they bother? Why do these programmers, who presumably could be well paid for doing something else, sit up nights to work on Apache for our benefit? There is no such thing as a free lunch, so they do it for a number of typically human reasons. One might list, in no particular order:

- They want to do something more interesting than their day job, which might be writing stock control packages for BigBins, Inc.
- They want to be involved on the edge of what is happening. Working on a project like this is a pretty good way to keep up-to-date. After that comes consultancy on the next hot project.
- The more worldly ones might remember how, back in the old days of 1995, quite a lot of the people working on the web server at NCSA left for a thing called Netscape and became, in the passage of the age, zillionaires.

- It's fun. Developing good software is interesting and amusing and you get to meet and work with other clever people.
- They are not doing the bit that programmers hate: explaining to end users why their treasure isn't working and trying to fix it in 10 minutes flat. If you want support on Apache you have to consult one of several commercial organizations (see Appendix A), who, quite properly, want to be paid for doing the work everyone loathes.

The Demonstration CD-ROM

The CD-ROM that accompanies this book can be read by both Win32 and Unix systems. It contains the requisite README file with installation instructions and other useful information. The CD-ROM contains Apache distributions for Unix and Windows and the demonstration web sites referred to throughout the book. The contents of the CD-ROM are organized into four directories:

distributions/
: This directory contains Apache and Cygwin distributions:

  - *apache_1.3.3.tar.gz* Apache 1.3.3 Unix distribution.
  - *apache_1_3_3.exe* Apache 1.3.3 Windows distribution.
  - *cygwin-b20/* directory Cygwin—Unix utilities for Windows.
    - — *readme.txt* Read this first!
    - — *user.exe* The (smaller) user distribution.
    - — *full.exe* The (larger) complete distribution.

install/
: This directory contains scripts to install the sample sites:

  - *install* Run this script to install the sites.
  - *install.conf* Unix configuration file for *install*.
  - *installwin.conf* Win32 configuration file for *install*.

sites/
: This directory contains the sample sites used in the book.

unpacked/
: This directory contains unpacked distributions:

  - *apache_1.3.3* Apache unpacked with *mod_reveal* added.

Conventions Used in This Book

This section covers the various conventions used in this book.

Typographic Conventions

`Constant Width`
: Used for HTTP headers, status codes, MIME content types, directives in configuration files, commands, options/switches, functions, methods, variable names, and code within body text

`Constant Width Bold`
: Used in code segments to indicate input to be typed in by the user

`Constant Width Italic`
: Used for replaceable items in code and text

Italic
: Used for filenames, pathnames, newsgroup names, Internet addresses (URLs), email addresses, variable names (except in examples), terms being introduced, program names, subroutine names, CGI script names, hostnames, usernames, and group names

Icons

UNIX Text marked with this icon applies to the Unix version of Apache.

WIN32 Text marked with this icon applies to the Win32 version of Apache.

The owl symbol designates a note relating to the surrounding text.

The turkey symbol designates a warning related to the surrounding text.

Pathnames

We use the text convention *.../* to indicate your path to the demonstration sites, which may well be different from ours. For instance, on our Apache machine, we kept all the demonstration sites in the directory */usr/www*. So, for example, our path would be */usr/www/site.simple*. You might want to keep the sites somewhere other than */usr/www,* so we refer to the path as *.../site.simple*.

Don't type .../ into your computer. The attempt will upset it!

Directives

Apache is controlled through roughly 150 directives. For each directive, a formal explanation is given in the following format:

Directive

```
Syntax
Where used
```

An explanation of the directive is located here.

So, for instance, we have the following directive:

ServerAdmin

```
ServerAdmin email address
Server config, virtual host
```

`ServerAdmin` gives the email address for correspondence. It automatically generates error messages so the user has someone to write to in case of problems.

The "where used" line explains the appropriate environment for the directive. This will become clearer later.

Organization of This Book

The chapters that follow and their contents are listed here:

Chapter 1, *Getting Started*
: Covers web servers, how Apache works, TCP/IP, HTTP, hostnames, what a client does, what happens at the server end, choosing a Unix version, and compiling and installing Apache under both Unix and Win32.

Chapter 2, *Our First Web Site*
: Discusses getting Apache to run, creating Apache users, runtime flags, permissions, and *site.simple*.

Chapter 3, *Toward a Real Web Site*
: Introduces a demonstration business, Butterthlies, Inc.; some HTML; default indexing of web pages; server housekeeping; and block directives.

Chapter 4, *Common Gateway Interface (CGI)*
: Demonstrates aliases, logs, HTML forms, shell script, a CGI in C, environment variables, and adapting to the client's browser.

Chapter 5, *Authentication*
: Explains controlling access, collecting information about clients, cookies, DBM control, digest authentication, and anonymous access.

Chapter 6, *MIME, Content and Language Negotiation*
: Covers content and language arbitration, type maps, and expiration of information.

Chapter 7, *Indexing*
: Discusses better indexes, index options, your own indexes, and imagemaps.

Chapter 8, *Redirection*
: Describes `Alias`, `ScriptAlias`, and the amazing `Rewrite` module.

Chapter 9, *Proxy Server*
: Covers remote proxies and proxy caching.

Chapter 10, *Server-Side Includes*
: Explains runtime commands in your HTML and XSSI—a more secure server-side include.

Chapter 11, *What's Going On?*
: Covers server status, logging the action, and configuring the log files.

Chapter 12, *Extra Modules*
: Discusses authentication, blocking, counters, faster CGI, languages, server-side scripting, and URL rewriting.

Chapter 13, *Security*
: Discusses Apache's security precautions, validating users, binary signatures, virtual cash, certificates, firewalls, packet filtering, secure sockets layer (SSL), legal issues, patent rights, national security, and Apache-SSL directives.

Chapter 14, *The Apache API*
: Describes pools; per-server, per-directory, and per-request information; functions; warnings; and parsing.

Chapter 15, *Writing Apache Modules*
: Covers status codes; module structure; the command table; the initializer, translate name, check access, check user ID, check authorization and check type routines; prerun fixups; handlers; the logger; and a complete example.

Appendix A, *Support Organizations*
: Provides a list of commercial service and/or consultation providers.

Appendix B, *The echo Program*
: Provides a listing of *echo.c*.

Appendix C, *NCSA and Apache Compatibility*
: Contains Apache Group internal mail discussing NCSA/Apache compatibility issues.

Appendix D, *SSL Protocol*
: Provides the SSL specification.

Appendix E, *Sample Apache Log*
Contains a listing of the full log file referenced in Chapter 11.

In addition, the Apache Quick Reference Card provides an outline of the Apache 1.3.4 syntax.

Acknowledgments

First, thanks to Robert S. Thau, who gave the world the Apache API and the code that implements it, and to the Apache Group, who worked on it before and have worked on it since. Thanks to Eric Young and Tim Hudson for giving SSLeay to the Web.

Thanks to Bryan Blank, Aram Mirzadeh, Chuck Murcko, and Randy Terbush, who read early drafts of the first edition text and made many useful suggestions; and to John Ackermann, Geoff Meek, and Shane Owenby, who did the same for the second edition. Thanks to Paul C. Kocher for allowing us to reproduce SSL Protocol, Version 3.0, in Appendix D, and to Netscape Corporation for allowing us to reproduce *echo.c* in Appendix B.

We would also like to offer special thanks to Andrew Ford for giving us permission to reprint his Apache Quick Reference Card.

Many thanks to Robert Denn, our editor at O'Reilly, who patiently turned our text into a book—again. The two layers of blunders that remain are our own contribution.

And finally, thanks to Camilla von Massenbach and Barbara Laurie, who have continued to put up with us while we rewrote this book.

1

Getting Started

When you connect to the URL of someone's home page—say the notional *http://www.butterthlies.com/* we shall meet later on—you send a message across the Internet to the machine at that address. That machine, you hope, is up and running, its Internet connection is working, and it is ready to receive and act on your message.

URL stands for Universal Resource Locator. A URL such as *http://www.butter-thlies.com/* comes in three parts:

```
<method>://<host>/<absolute path URL (apURL)>
```

So, in our example, *<method>* is `http`, meaning that the browser should use HTTP (Hypertext Transfer Protocol); *<host>* is `www.butterthlies.com`; and *<apURL>* is "`/`", meaning the top directory of the host. Using HTTP/1.1, your browser might send the following request:

```
GET / HTTP/1.1
Host: www.butterthlies.com
```

The request arrives at port 80 (the default HTTP port) on the host *www.butterthlies.com*. The message is again in three parts: a method (an HTTP method, not a URL method), that in this case is `GET`, but could equally be `PUT`, `POST`, `DELETE`, or `CONNECT`; the Uniform Resource Identifier (URI) "`/`"; and the version of the protocol we are using. It is then up to the web server running on that host to make something of this message.

It is worth saying here—and we will say it again—that the whole business of a web server is to translate a URL either into a filename, and then send that file back over the Internet, or into a program name, and then run that program and send its output back. That is the meat of what it does: all the rest is trimming.

The host machine may be a whole cluster of hypercomputers costing an oil sheik's ransom, or a humble PC. In either case, it had better be running a web server, a program that listens to the network and accepts and acts on this sort of message.

What do we want a web server to do? It should:

- Run fast, so it can cope with a lot of inquiries using a minimum of hardware.
- Be multitasking, so it can deal with more than one inquiry at once.
- Be multitasking, so that the person running it can maintain the data it hands out without having to shut the service down. Multitasking is hard to arrange within a program: the only way to do it properly is to run the server on a multitasking operating system. In Apache's case, this is some flavor of Unix (or Unix-like system), Win32, or OS/2.
- Authenticate inquirers: some may be entitled to more services than others. When we come to virtual cash, this feature (see Chapter 13, *Security*) becomes essential.
- Respond to errors in the messages it gets with answers that make sense in the context of what is going on. For instance, if a client requests a page that the server cannot find, the server should respond with a "404" error, which is defined by the HTTP specification to mean "page does not exist."
- Negotiate a style and language of response with the inquirer. For instance, it should—if the people running the server can rise to the challenge—be able to respond in the language of the inquirer's choice. This ability, of course, can open up your site to a lot more action. And there are parts of the world where a response in the wrong language can be a bad thing. If you were operating in Canada, where the English/French divide arouses bitter feelings, or in Belgium, where the French/Flemish split is as bad, this feature could make or break your business.
- Offer different formats. On a more technical level, a user might want JPEG image files rather than GIF, or TIFF rather than either of the former. He or she might want text in vdi format rather than PostScript.
- Run as a proxy server. A proxy server accepts requests for clients, forwards them to the real servers, and then sends the real servers' responses back to the clients. There are two reasons why you might want a proxy server:
  - The proxy might be running on the far side of a firewall (see Chapter 13), giving its users access to the Internet.
  - The proxy might cache popular pages to save reaccessing them.

- Be secure. The Internet world is like the real world, peopled by a lot of lambs and a few wolves.* The wolves like to get into the lambs' folds (of which your computer is one) and, when there, raven and tear in the usual wolfish way. The aim of a good server is to prevent this happening. The subject of security is so important that we will come back to it several times before we are through.

These are services that the developers of Apache think a server should offer. There are people who have other ideas, and, as with all software development, there are lots of features that might be nice—features someone might use one day, or that might, if put into the code, actually make it work better instead of fouling up something else that has, until then, worked fine. Unless developers are careful, good software attracts so many improvements that it eventually rolls over and sinks like a ship caught in an Arctic ice storm.

Some ideas are in progress: in particular, various proposals for Apache 2.0 are being kicked around. The main features Apache 2.0 is supposed to have are multithreading (on platforms that support it), layered I/O, and a rationalized API.

If you have bugs to report or more ideas for development, look at *http://www.apache.org/bug_report.html*. You can also try *news:comp.infosystems.www.servers.unix*, where some of the Apache team lurk, along with many other knowledgeable people, and *news:comp.infosystems.www.servers.ms-windows*.

How Does Apache Work?

Apache is a program that runs under a suitable multitasking operating system. In the examples in this book, the operating systems are Unix and Windows 95/98/NT, which we call *Win32*. The binary is called *httpd* under Unix and *apache.exe* under Win32† and normally runs in the background. Each copy of *httpd/apache* that is started has its attention directed at a *web site*, which is, for practical purposes, a directory. For an example, look at *site.toddle* on the demonstration CD-ROM. Regardless of operating system, a site directory typically contains four subdirectories:

conf
: Contains the configuration file(s), of which *httpd.conf* is the most important. It is referred to throughout this book as the *Config* file.

* We generally follow the convention of calling these people the Bad Guys. This avoids debate about "hackers," which, to many people, simply refers to good programmers, but to some means Bad Guys. We discover from the French edition of this book that in France they are *Sales Types*—dirty fellows.

† This double name is rather annoying, but it seems that life has progressed too far for anything to be done about it. We will, rather clumsily, refer to *httpd/apache* and hope that the reader can pick the right one.

htdocs
: Contains the HTML scripts to be served up to the site's clients. This directory and those below it, the *web space*, are accessible to anyone on the Web and therefore pose a severe security risk if used for anything other than public data.

logs
: Contains the log data, both of accesses and errors.

cgi-bin
: Contains the CGI scripts. These are programs or shell scripts written by or for the webmaster that can be executed by Apache on behalf of its clients. It is most important, for security reasons, that this directory not be in the web space.

In its idling state, Apache does nothing but listen to the IP addresses and TCP port or ports specified in its Config file. When a request appears on a valid port, Apache receives the HTTP request and analyzes the headers. It then applies the rules it finds in the Config file and takes the appropriate action.

The webmaster's main control over Apache is through the Config file. The webmaster has some 150 *directives* at his or her disposal; most of this book is an account of what these directives do and how to use them to reasonable advantage. The webmaster also has half a dozen flags he or she can use when Apache starts up. Apache is *freeware*: the intending user downloads the source code and compiles it (under Unix) or downloads the executable (for Windows) from *www.apache.org* or a suitable mirror site. You can also load the source code from the demonstration CD-ROM included with this book, although it is not the most recent. Although it sounds like a difficult business to download the source code and configure and compile it, it only takes about 20 minutes and is well worth the trouble.

UNIX Under Unix, the webmaster also controls which *modules* are compiled into Apache. Each module provides the code to execute a number of directives. If there is a group of directives that aren't needed, the appropriate modules can be left out of the binary by commenting their names out in the *configuration file*[*] that controls the compilation of the Apache sources. Discarding unwanted modules reduces the size of the binary and may improve performance.

WIN32 Under Windows, Apache is normally precompiled as an executable. The core modules are compiled in, and others are loaded, if needed, as dynamic link librar-

[*] It is important to distinguish between the configuration file used at compile time and the Config file used to control the operation of a web site.

ies (DLLs) at runtime, so control of the executable's size is less urgent. The DLLs supplied in the *.../apache/modules* subdirectory are as follows:

```
APACHE~1 DLL         5,120  19/07/98  11:47 ApacheModuleAuthAnon.dll
APACHE~2 DLL         5,632  19/07/98  11:48 ApacheModuleCERNMeta.dll
APACHE~3 DLL         6,656  19/07/98  11:47 ApacheModuleDigest.dll
APACHE~4 DLL         6,144  19/07/98  11:48 ApacheModuleExpires.dll
APACHE~5 DLL         5,120  19/07/98  11:48 ApacheModuleHeaders.dll
APACHE~6 DLL        46,080  19/07/98  11:48 ApacheModuleProxy.dll
APACHE~7 DLL        35,328  19/07/98  11:48 ApacheModuleRewrite.dll
APACHE~8 DLL         6,656  19/07/98  11:48 ApacheModuleSpeling.dll
APACHE~9 DLL        10,752  19/07/98  11:47 ApacheModuleStatus.dll
APACH~10 DLL         6,144  19/07/98  11:48 ApacheModuleUserTrack.dll
```

What these are and what they do will become more apparent as we proceed. You can add other DLLs from outside suppliers; more will doubtless become available.

It is also possible to download the source code and compile it for Win32 using Microsoft Visual C++ v5.0. We describe this in ""Apache Under Windows," later in this chapter. You might do this if you wanted to write your own module (see Chapter 15, *Writing Apache Modules*).

What to Know About TCP/IP

To understand the substance of this book, you need a modest knowledge of what TCP/IP is and what it does. You'll find more than enough information in Craig Hunt and Robert Bruce Thompson's books on TCP/IP,* but what follows is, we think, what is necessary to know for our book's purposes.

TCP/IP (Transmission Control Protocol/Internet Protocol) is a set of protocols enabling computers to talk to each other over networks. The two protocols that give the suite its name are among the most important, but there are many others, and we shall meet some of them later. These protocols are embodied in programs on your computer written by someone or other; it doesn't much matter who. TCP/IP seems unusual among computer standards in that the programs that implement it actually work, and their authors have not tried too much to improve on the original conceptions.

TCP/IP only applies where there is a network. Each computer on a network that wants to use TCP/IP has an *IP address*, for example, 192.168.123.1.

There are four parts in the address, separated by periods. Each part corresponds to a byte, so the whole address is four bytes long. You will, in consequence, seldom see any of the parts outside the range 0–255.

* *Windows NT TCP/IP Network Administration*, by Craig Hunt and Robert Bruce Thompson (O'Reilly & Associates), and *TCP/IP Network Administration, Second Edition*, by Craig Hunt (O'Reilly & Associates).

Although not required by protocol, by convention there is a dividing line somewhere inside this number: to the left is the network number and to the right, the host number. Two machines on the same physical network—usually a local area network (LAN)—normally have the same network number and communicate directly using TCP/IP.

How do we know where the dividing line is between network number and host number? The default dividing line is determined by the first of the four numbers: if the value of the first number is:

- 0–127 (first byte is 0xxxxxxx binary), the dividing line is after the first number, and it is a Class A network. There are few class A networks—125 usable ones—but each one supports up to 16,777,214 hosts.
- 128–191 (first byte is 10xxxxxx binary), the dividing line is after the second number, and it is a Class B network. There are more class B networks—16,382—and each one can support up to 65,534 hosts.
- 192–223 (first byte is 110xxxxx binary), the dividing line is after the third number, and it is a Class C network. There is a huge number of class C networks—2,097,150—but each one supports a paltry 254 hosts.

The remaining values of the first number, 224–255, are not relevant here. Network numbers—the left-hand part—that are all 0s* or all 1s† in binary are reserved and therefore not relevant to us either. These addresses are as follows:

- 0.x.x.x
- 127.x.x.x
- 128.0.x.x
- 191.255.x.x
- 192.0.0.x
- 223.255.255.x

It is often possible to bypass the rules of Class A, B, and C networks using *subnet masks*. These allow us to further subdivide the network by using more of the bits for the network number and less for the host number. Their correct use is rather technical, so we leave it to the experts.

You do not need to know this information in order to run a host, because the numbers you deal with are assigned to you by your network administrator or are

* An all-0 network address means "this network." This is defined in STD 5 (RFC 791).

† An all-1 network address means "broadcast." This is also defined in STD 5 (RFC 922). In practice, broadcast network addresses are not very useful, and, indeed, some of these "reserved" addresses have already been used for other purposes; for example, 127.0.0.1 means "this machine," by convention.

just facts of the Internet. But we feel you should have some understanding in order to avoid silly conversations with people who do know about TCP/IP. It is also relevant to virtual hosting because each virtual host (see Chapter 3, *Toward a Real Web Site*) must have its own IP address (at least until HTTP/1.1 is in wide use).

Now we can think about how two machines with IP addresses X and Y talk to each other. If X and Y are on the same network, and are correctly configured so that they have the same network number and different host numbers, they should be able to fire up TCP/IP and send packets to each other down their local, physical network without any further ado.

If the network numbers are not the same, TCP/IP sends the packets to a *router*, a special machine able, by processes that do not concern us here, to find out where the other machine is and deliver the packets to it. This communication may be over the Internet or might occur on your wide area network (WAN).

There are two ways computers use TCP/IP to communicate:

UDP (User Datagram Protocol)
: A way to send a single packet from one machine to another. It does not guarantee delivery, and there is no acknowledgment of receipt. It is nasty for our purposes, and we don't use it.

TCP (Transmission Control Protocol)
: A way to establish communications between two computers. It reliably delivers messages of any size. This is a better protocol for our purposes.

How Does Apache Use TCP/IP?

Let's look at a server from the outside. We have a box in which there is a computer, software, and a connection to the outside world—a piece of Ethernet or a serial line to a modem, for example. This connection is known as an *interface* and is known to the world by its IP address. If the box had two interfaces, they would each have an IP address, and these addresses would normally be different. One interface, on the other hand, may have more than one IP address (see Chapter 3).

Requests arrive on an interface for a number of different services offered by the server using different protocols:

- Network News Transfer Protocol (NNTP): news
- Simple Mail Transfer Protocol (SMTP): mail
- Domain Name Service (DNS)
- HTTP: World Wide Web

The server can decide how to handle these different requests because the four-byte IP address that leads the request to its interface is followed by a two-byte port number. Different services attach to different ports:

- NNTP: port number 119
- SMTP: port number 25
- DNS: port number 53
- HTTP: port number 80

As the local administrator or webmaster, you can (if you really want) decide to attach any service to any port. Of course, if you decide to step outside convention, you need to make sure that your clients share your thinking. Our concern here is just with WWW and Apache. Apache, by default, listens to port number 80 because it deals in WWW business.

UNIX Port numbers below 1024 can only be used by the superuser (*root*, under Unix); this prevents other users from running programs masquerading as standard services, but brings its own problems, as we shall see.

WIN32 Under Win32 there is currently no real security beyond what you can provide yourself (using file permissions) and no superuser (at least, not as far as port numbers are concerned).

This is fine if our machine is providing only one web server to the world. In real life, you may want to host several, many, dozens, or even hundreds of servers, which appear to the world to be completely different from each other. This situation was not anticipated by the authors of HTTP/1.0, so handling a number of hosts on one machine has to be done by a kludge, which is to assign multiple addresses to the same interface and distinguish the virtual host by its IP address. This technique is known as *IP-intensive virtual hosting*. Using HTTP/1.1, virtual hosts may be created by assigning multiple names to the same IP address. The browser sends a `Host` header to say which name it is using.

Multiple Sites: Unix

By happy accident, the crucial Unix utility *ifconfig*, which binds IP addresses to physical interfaces, often allows the binding of multiple IP numbers so that people can switch from one IP number to another and maintain service during the transition.

In practical terms, on many versions of Unix, we run *ifconfig* to give multiple IP addresses to the same interface. The interface in this context is actually the bit of software—the driver—that handles the physical connection (Ethernet card, serial