Table of Contents

Java Network Programming, 3rd Edition By Elliotte Rusty Harold ISBN: 0596007213 Publisher: O'Reilly
Print Publication Date: 10/1/2004

Chapter 2. Basic Network Concepts

This chapter covers the background networking concepts you need to understand before writing networked programs in Java (or, for that matter, in any language). Moving from the most general to the most specific, it explains what you need to know about networks in general, IP and TCP/IP-based networks in particular, and the Internet. This chapter doesn't try to teach you how to wire a network or configure a router, but you will learn what you need to know to write applications that communicate across the Internet. Topics covered in this chapter include the definition of network, the TCP/IP layer model, the IP, TCP, and UDP protocols, firewalls and proxy servers, the Internet, and the Internet standardization process. Experienced network gurus may safely skip this chapter.

2.1. Networks

A *network* is a collection of computers and other devices that can send data to and receive data from each other, more or less in real time. A network is often connected by wires, and the bits of data are turned into electromagnetic waves that move through the wires. However, wireless networks transmit data through infrared light and microwaves, and many long-distance transmissions are now carried over fiber optic cables that send visible light through glass filaments. There's nothing sacred about any particular physical medium for the transmission of data. Theoretically, data could be transmitted by coal-powered computers that send smoke signals to each other. The response time (and environmental impact) of such a network would be rather poor.

Each machine on a network is called a *node*. Most nodes are computers, but printers, routers, bridges, gateways, dumb terminals, and Coca-Cola™ machines can also be nodes. You might use Java to interface with a Coke machine but otherwise, you'll mostly talk to other computers. Nodes that are fully functional computers are also called *hosts*. We will use the word *node* to refer to any device on the network, and the word *host* to refer to a node that is a general-purpose computer.

Every network node has an *address*, a series of bytes that uniquely identify it. You can think of this group of bytes as a number, but in general the number of bytes in an address or the ordering of those bytes (big endian or little endian) is not guaranteed to match any primitive numeric data type in Java. The more bytes there are in each address, the more addresses there are available and the more devices that can be connected to the network simultaneously.

Addresses are assigned differently on different kinds of networks. AppleTalk addresses are chosen randomly at startup by each host. The host then checks to see if any other machine on the network is using that address. If another machine is using the address, the host randomly chooses another, checks to see if that address is already in use, and so on until it gets one that isn't being used. Ethernet addresses are attached to the physical Ethernet hardware. Manufacturers of Ethernet hardware use pre-assigned manufacturer codes to make sure there are no conflicts between the addresses in their hardware and the addresses of other manufacturer's hardware. Each manufacturer is responsible for making sure it doesn't ship two Ethernet cards with the same address. Internet addresses are normally assigned to a computer by the organization that is responsible for it. However, the addresses that an organization is allowed to choose for its computers are assigned by the organization's Internet Service Provider (ISP). ISPs get their IP addresses from one of four regional Internet Registries (the registry for North America is ARIN, the American Registry for Internet Numbers, at http://www.arin.net/), which are in turn assigned IP addresses by the Internet Corporation for Assigned Names and Numbers (ICANN, at http://www.icann.org/).

On some kinds of networks, nodes also have names that help human beings identify them. At a set moment in time, a particular name normally refers to exactly one address. However, names are not locked to addresses. Names can change while addresses stay the same or addresses can change while the names stay the same. It is not uncommon for one address to have several names and it is possible, though somewhat less common, for one name to refer to several different addresses.

All modern computer networks are *packet-switched* networks: data traveling on the network is broken into chunks called *packets* and each packet is handled separately. Each packet contains information about who sent it and where it's going. The most important advantage of breaking data into individually addressed packets is that packets from many ongoing exchanges can travel on one wire, which makes it much cheaper to build a network: many computers can share the same wire without interfering. (In contrast, when you make a local telephone call within the same exchange, you have essentially reserved a wire from your phone to the phone of the person you're calling. When all the wires are in use, as sometimes happens during a major emergency or holiday, not everyone who picks up a phone will get a dial tone. If you stay on the line, you'll eventually get a dial tone when a line becomes free. In some countries with worse phone service than the United States, it's not uncommon to have to wait half an hour or more for a dial tone.) Another advantage of packets is that checksums can be used to detect whether a packet was damaged in transit.

We're still missing one important piece: some notion of what computers need to say to pass data back and forth. A *protocol* is a precise set of rules defining how computers communicate: the format of addresses, how data is split into packets, and so on. There are many different protocols defining different aspects of network communication. For example, the Hypertext Transfer Protocol (HTTP) defines how web browsers and servers communicate; at the other end of the spectrum, the IEEE 802.3 standard defines a protocol for how bits are encoded as electrical signals on a particular type of wire (among other protocols). Open, published protocol standards allow software and equipment from different vendors to communicate with each other: your web browser doesn't care whether any given server is a Unix workstation, a Windows box, or a Macintosh, because the server and the browser speak the same HTTP protocol regardless of platform.

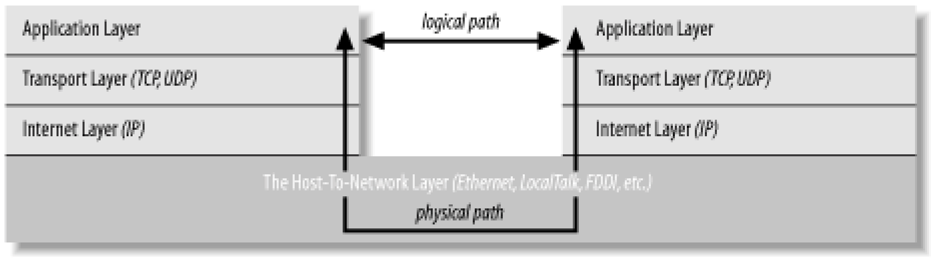
2.2. The Layers of a Network

Sending data across a network is a complex operation that must be carefully tuned to the physical characteristics of the network as well as the logical character of the data being sent. Software that sends data across a network must understand how to avoid collisions between packets, convert digital data to analog signals, detect and correct errors, route packets from one host to another, and more. The process becomes even more complicated when the requirement to support multiple operating systems and heterogeneous network cabling is added.

To make this complexity manageable and hide most of it from the application developer and end user, the different aspects of network communication are separated into multiple layers. Each layer represents a different level of abstraction between the physical hardware (e.g., the wires and electricity) and the information being transmitted. Each layer has a strictly limited function. For instance, one layer may be responsible for routing packets, while the layer above it is responsible for detecting and requesting retransmission of corrupted packets. In theory, each layer only talks to the layers immediately above and immediately below it. Separating the network into layers lets you modify or even replace the software in one layer without affecting the others, as long as the interfaces between the layers stay the same.

There are several different layer models, each organized to fit the needs of a particular kind of network. This book uses the standard TCP/IP four-layer model appropriate for the Internet, shown in Figure 2-1. In this model, applications like Internet Explorer and Eudora run in the application layer and talk only to the transport layer. The transport layer talks only to the application layer and the internet layer. The internet layer in turn talks only to the host-to-network layer and the transport layer, never directly to the application layer. The host-to-network layer moves the data across the wires, fiber optic cables, or other medium to the host-to-network layer on the remote system, which then moves the data up the layers to the application on the remote system.

Figure 2-1. The layers of a network


Application Layer
Transport Layer (TCP, UDP)
Internet Layer (IP)
logical path
Application Layer
Transport Layer (TCP, UDP)
Internet Layer (IP)
The Host-To-Network Layer (Ethernet, LocalTalk, FDDI, etc.)
physical path

For example, when a web browser sends a request to a web server to retrieve a page, the browser is actually only talking to the transport layer on the local client machine. The transport layer breaks the request up into TCP segments, adds some sequence numbers and checksums to the data, and then passes the request to the local internet layer. The internet layer fragments the segments into IP datagrams of the necessary size for the local network and passes them to the host-to-network layer for transmission onto the wire. The host-to-network layer encodes the digital data as analog signals appropriate for the particular physical medium and sends the request out the wire where it will be read by the host-to-network layer of the remote system to which it's addressed.

The host-to-network layer on the remote system decodes the analog signals into digital data then passes the resulting IP datagrams to the server's internet layer. The internet layer does some simple checks to see that the IP datagrams aren't corrupt, reassembles them if they've been fragmented, and passes them to the server's transport layer. The server's transport layer checks to see that all the data arrived and requests retransmission of any missing or corrupt pieces. (This request actually goes back down through the server's internet layer, through the server's host-to-network layer, and back to the client system, where it bubbles back up to the client's transport layer, which retransmits the missing data back down through the layers. This is all transparent to the application layer.) Once the server's transport layer has received enough contiguous, sequential datagrams, it reassembles them and writes them onto a stream read by the web server running in the server application layer. The server responds to the request and sends its response back down through the layers on the server system for transmission back across the Internet and delivery to the web client.

As you can guess, the real process is much more elaborate. The host-to-network layer is by far the most complex, and a lot has been deliberately hidden. For example, it's entirely possible that data sent across the Internet will pass through several routers and their layers before reaching its final destination. However, 90% of the time your Java code will work in the application layer and only need to talk to the transport layer. The other 10% of the time, you'll be in the transport layer and talking to the application layer or the internet layer. The complexity of the host-to-network layer is hidden from you; that's the point of the layer model.

If you read the network literature, you're likely to encounter an alternative seven-layer model called the Open Systems Interconnection Reference Model (OSI). For network programs in Java, the OSI model is overkill. The biggest difference between the OSI model and the TCP/IP model used in this book is that the OSI model splits the host-to-network layer into data link and physical layers and inserts presentation and session layers in between the application and transport layers. The OSI model is more general and better suited for non-TCP/IP networks, although most of the time it's still overly complex. In any case, Java's network classes only work on TCP/IP networks and always in the application or transport layers, so for the purposes of this book, absolutely nothing is gained by using the more complicated OSI model.

To the application layer, it seems as if it is talking directly to the application layer on the other system; the network creates a logical path between the two application layers. It's easy to understand the logical path if you think about an IRC chat session. Most participants in an IRC chat would say that they're talking to another person. If you really push them, they might say that they're talking to their computer (really the application layer), which is talking to the other person's computer, which is talking to the other person. Everything more than one layer deep is effectively invisible, and that is exactly the way it should be. Let's consider each layer in more detail.

2.2.1. The Host-to-Network Layer

As a Java programmer, you're fairly high up in the network food chain. A lot happens below your radar. In the standard reference model for IP-based Internets (the only kind of network Java really understands), the hidden parts of the network belong to the *host-to-network layer* (also known as the link layer, data link layer, or network interface layer). The host-to-network layer defines how a particular network interface—such as an Ethernet card or a PPP connection—sends IP datagrams over its physical connection to the local network and the world.

The part of the host-to-network layer made up of the hardware that connects different computers (wires, fiber optic cables, microwave relays, or smoke signals) is sometimes called the physical layer of the network. As a Java programmer, you don't need to worry about this layer unless something goes wrong—the plug falls out of the back of your computer, or someone drops a backhoe through the T-1 line between you and the rest of the world. In other words, Java never sees the physical layer.

For computers to communicate with each other, it isn't sufficient to run wires between them and send electrical signals back and forth. The computers have to agree on certain standards for how those signals are interpreted. The first step is to determine how the packets of electricity or light or smoke map into bits and bytes of data. Since the physical layer is analog, and bits and bytes are digital, this process involves a digital-to-analog conversion on the sending end and an analog-to-digital conversion on the receiving end.

Since all real analog systems have noise, error correction and redundancy need to be built into the way data is translated into electricity. This is done in the data link layer. The most common data link layer is Ethernet. Other popular data link layers include TokenRing, PPP, and Wireless Ethernet (802.11). A specific data link layer requires specialized hardware. Ethernet cards won't communicate on a TokenRing network, for example. Special devices called *gateways* convert information from one type of data link layer, such as Ethernet, to another, such as TokenRing. As a Java programmer, the data link layer does not affect you directly. However, you can sometimes optimize the data you send in the application layer to match the native packet size of a particular data link layer, which can have some affect on performance. This is similar to matching disk reads and writes to the native block size of the disk. Whatever size you choose, the program will still run, but some sizes let the program run more efficiently than others, and which sizes these are can vary from one computer to the next.

2.2.2. The Internet Layer

The next layer of the network, and the first that you need to concern yourself with, is the *internet layer*. In the OSI model, the internet layer goes by the more generic name *network layer*. A network layer protocol defines how bits and bytes of data are organized into the larger groups called packets, and the addressing scheme by which different machines find each other. The Internet Protocol (IP) is the most widely used network layer protocol in the world and the only network layer protocol Java understands. IP is almost exclusively the focus of this book. Other, semi-common network layer protocols include Novell's IPX, and IBM and

Microsoft's NetBEUI, although nowadays most installations have replaced these protocols with IP. Each network layer protocol is independent of the lower layers. IP, IPX, NetBEUI, and other protocols can each be used on Ethernet, Token Ring, and other data link layer protocol networks, each of which can themselves run across different kinds of physical layers.

Data is sent across the internet layer in packets called *datagrams*. Each IP datagram contains a header between 20 and 60 bytes long and a payload that contains up to 65,515 bytes of data. (In practice, most IP datagrams are much smaller, ranging from a few dozen bytes to a little more than eight kilobytes.) The header of each IP datagram contains these items, in this order:

4-bit version number
: Always 0100 (decimal 4) for current IP; will be changed to 0110 (decimal 6) for IPv6, but the entire header format will also change in IPv6.

4-bit header length
: An unsigned integer between 0 and 15 specifying the number of 4-byte words in the header; since the maximum value of the header length field is 1111 (decimal 15), an IP header can be at most 60 bytes long.

1-byte type of service
: A 3-bit precedence field that is no longer used, four type-of-service bits (minimize delay, maximize throughput, maximize reliability, minimize monetary cost) and a zero bit. Not all service types are compatible. Many computers and routers simply ignore these bits.

2-byte datagram length
: An unsigned integer specifying the length of the entire datagram, including both header and payload.

2-byte identification number
: A unique identifier for each datagram sent by a host; allows duplicate datagrams to be detected and thrown away.

3-bit flags
: The first bit is 0; the second bit is 0 if this datagram may be fragmented, 1 if it may not be; and the third bit is 0 if this is the last fragment of the datagram, 1 if there are more fragments.

13-bit fragment offset
: In the event that the original IP datagram is fragmented into multiple pieces, this field identifies the position of this fragment in the original datagram.

1-byte time-to-live (TTL)
: Number of nodes through which the datagram can pass before being discarded; used to avoid infinite loops.

1-byte protocol
: 6 for TCP, 17 for UDP, or a different number between 0 and 255 for each of more than 100 different protocols (some quite obscure); see http://www.iana.org/assignments/protocol-numbers for the complete current list.

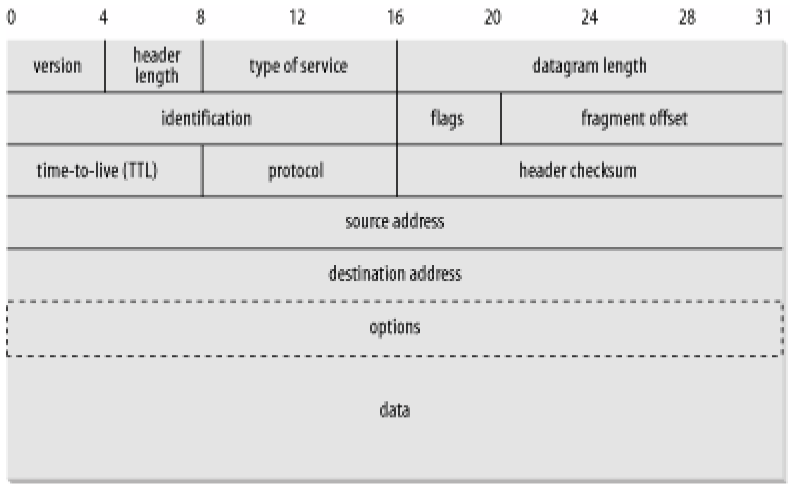
2-byte header checksum
: A checksum of the header only (not the entire datagram) calculated using a 16-bit one's complement sum.

4-byte source address
: The IP address of the sending node.

4-byte destination address
: The IP address of the destination node.

In addition, an IP datagram header may contain between 0 and 40 bytes of optional information, used for security options, routing records, timestamps, and other features Java does not support. Consequently, we will not discuss them here. The interested reader is referred to *TCP/IP Illustrated, Volume 1: The Protocols*, by W. Richard Stevens (Addison Wesley), for more details on these fields. Figure 2-2 shows how the different quantities are arranged in an IP datagram. All bits and bytes are big-endian; most significant to least significant runs left to right.

Figure 2-2. The structure of an IPv4 datagram



2.2.3. The Transport Layer

Raw datagrams have some drawbacks. Most notably, there's no guarantee that they will be delivered. Even if they are delivered, they may have been corrupted in transit. The header checksum can only detect corruption in the header, not in the data portion of a datagram. Finally, even if the datagrams arrive uncorrupted, they do not necessarily arrive in the order in which they were sent. Individual datagrams may follow different routes from source to destination. Just because datagram A is sent before datagram B does not mean that datagram A will arrive before datagram B.

The *transport layer* is responsible for ensuring that packets are received in the order they were sent and making sure that no data is lost or corrupted. If a packet is lost, the transport layer can ask the sender to retransmit the packet. IP networks implement this by adding an additional header to each datagram that contains more information. There are two primary protocols at this level. The first, the Transmission Control Protocol (TCP), is a high-overhead protocol that allows for retransmission of lost or corrupted data and delivery of bytes in the order they were sent. The second protocol, the User Datagram Protocol (UDP), allows the receiver to detect corrupted packets but does not guarantee that packets are delivered in the correct order (or at all). However, UDP is often much faster than TCP. TCP is called a *reliable* protocol; UDP is an *unreliable* protocol. Later, we'll see that unreliable protocols are much more useful than they sound.

2.2.4. The Application Layer

The layer that delivers data to the user is called the *application layer*. The three lower layers all work together to define how data is transferred from one computer to another. The application layer decides what to do with the data after it's transferred. For example, an application protocol like HTTP (for the World Wide Web) makes sure that your web browser knows to display a graphic image as a picture, not a long stream of numbers. The application layer is where most of the network parts of your programs spend their time. There is an entire alphabet soup of application layer protocols; in addition to HTTP for the Web, there are SMTP, POP, and IMAP for email; FTP, FSP, and TFTP for file transfer; NFS for file access; NNTP for news transfer; Gnutella, FastTrack, and Freenet for file sharing; and many, many more. In addition, your programs can define their own application layer protocols as necessary.

2.3. IP, TCP, and UDP

IP, the Internet protocol, has a number of advantages over competing protocols such as AppleTalk and IPX, most stemming from its history. It was developed with military sponsorship during the Cold War, and ended up with a lot of features that the military was interested in. First, it had to be robust. The entire network couldn't stop functioning if the Soviets nuked a router in Cleveland; all messages still had to get through to their intended destinations (except those going to Cleveland, of course). Therefore IP was designed to allow multiple routes between any two points and to route packets of data around damaged routers.

Second, the military had many different kinds of computers, and all of them had to be able to talk to each other. Therefore the IP had to be open and platform-independent; it wasn't good enough to have one protocol for IBM mainframes and another for PDP-11s. The IBM mainframes needed to talk to the PDP-11s and any other strange computers that might be lying around.

Since there are multiple routes between two points, and since the quickest path between two points may change over time as a function of network traffic and other factors (such as the existence of Cleveland), the packets that make up a particular data stream may not all take the same route. Furthermore, they may not arrive in the order they were sent, if they even arrive at all. To improve on the basic scheme, TCP was layered on top of IP to give each end of a connection the ability to acknowledge receipt of IP packets and request retransmission of lost or corrupted packets. Furthermore, TCP allows the packets to be put back together on the receiving end in the same order they were sent.

TCP, however, carries a fair amount of overhead. Therefore, if the order of the data isn't particularly important and if the loss of individual packets won't completely corrupt the data stream, packets are sometimes sent without the guarantees that TCP provides. This is accomplished through the use of the UDP protocol. UDP is an unreliable protocol that does not guarantee that packets will arrive at their destination or that they will arrive in the same order they were sent. Although this would be a problem for uses such as file transfer, it is perfectly acceptable for applications where the loss of some data would go unnoticed by the end user. For example, losing a few bits from a video or audio signal won't cause much degradation; it would be a bigger problem if you had to wait for a protocol like TCP to request a retransmission of missing data. Furthermore, error-correcting codes can be built into UDP data streams at the application level to account for missing data.

A number of other protocols can run on top of IP. The most commonly requested is ICMP, the Internet Control Message Protocol, which uses raw IP datagrams to relay error messages between hosts. The best-known use of this protocol is in the ping program. Java does not support ICMP nor does it allow the sending of raw IP datagrams (as opposed to TCP segments or UDP datagrams). The only protocols Java supports are TCP and UDP, and application layer protocols built on top of these. All other transport layer, internet layer, and lower layer protocols such as ICMP, IGMP, ARP, RARP, RSVP, and others can only be implemented in Java programs by using native code.

2.3.1. IP Addresses and Domain Names

As a Java programmer, you don't need to worry about the inner workings of IP, but you do need to know about addressing. Every computer on an IPv4 network is identified by a four-byte number. This is normally written in a *dotted quad* format like 199.1.32.90, where each of the four numbers is one unsigned byte ranging in value from 0 to 255. Every computer attached to an IPv4 network has a unique four-byte address. When data is transmitted across the network, the packet's header includes the address of the machine for which the packet is intended (the destination address) and the address of the machine that sent the packet (the source address). Routers along the way choose the best route to send the packet along by inspecting the destination address. The source address is included so the recipient will know who to reply to.

There are a little more than four billion possible IP addresses, not even one for every person on the planet, much less for every computer. To make matters worse, the addresses aren't allocated very efficiently. A slow transition is under way to IPv6, which will use 16-byte addresses. This provides enough IP addresses to identify every person, every computer, and indeed every atom on the planet. IPv6 addresses are customarily written in eight blocks of four hexadecimal digits separated by colons, such as

FEDC:BA98:7654:3210:FEDC:BA98:7654:3210. Leading zeros do not need to be written. A double colon, at most one of which may appear in any address, indicates multiple zero blocks. For example, *FEDC:0000:0000:0000:00DC:0000:7076:0010* could be written more compactly as *FEDC::DC:0:7076:10*. In mixed networks of IPv6 and IPv4, the last four bytes of the IPv6 address are sometimes written as an IPv4 dotted quad address. For example, *FEDC:BA98:7654:3210:FEDC:BA98:7654:3210* could be written as *FEDC:BA98:7654:3210:FEDC:BA98:118.84.50.16*. IPv6 is only supported in Java 1.4 and later. Java 1.3 and earlier only support four-byte addresses.

Although computers are very comfortable with numbers, human beings aren't very good at remembering them. Therefore the Domain Name System (DNS) was developed to translate hostnames that humans can remember (like www.oreilly.com) into numeric Internet addresses (like 208.201.239.37). When Java programs access the network, they need to process both these numeric addresses and their corresponding hostnames. Methods for doing this are provided by the `java.net.InetAddress` class, which is discussed in Chapter 6.

Some computers, especially servers, have fixed addresses. Others, especially clients on local area networks and dial-up connections, receive a different address every time they boot up, often provided by a DHCP server or a PPP server. This is not especially relevant to your Java programs. Mostly you just need to remember that IP addresses may change over time, and not write any code that relies on a system having the same IP address. For instance, don't serialize the local IP address when saving application state. Instead, look it up fresh each time your program starts. It's also possible, although less likely, for an IP address to change while the program is running (for instance, if a dialup connection hangs up and then reconnects), so you may want to check the current IP address every time you need it rather than caching it. Otherwise, the difference between a dynamically and manually assigned address is not significant to Java programs.

2.3.2. Ports

Addresses would be all you needed if each computer did no more than one thing at a time. However, modern computers do many different things at once. Email needs to be separated from FTP requests, which need to be separated from web traffic. This is accomplished through *ports*. Each computer with an IP address has several thousand logical ports (65,535 per transport layer protocol, to be precise). These are purely abstractions in the computer's memory and do not represent anything physical, like a serial or parallel port. Each port is identified by a number between 1 and 65,535. Each port can be allocated to a particular service.

For example, HTTP, the underlying protocol of the Web, generally uses port 80. We say that a web server *listens* on port 80 for incoming connections. When data is sent to a web server on a particular machine at a particular IP address, it is also sent to a particular port (usually port 80) on that machine. The receiver checks each packet it sees for the port and sends the data to any programs that are listening to the specified port. This is how different types of traffic are sorted out.

Port numbers between 1 and 1,023 are reserved for well-known services like finger, FTP, HTTP, and IMAP. On Unix systems, including Linux and Mac OS X, only programs running as root can receive data from these ports, but all programs may send data to them. On Windows and Mac OS 9, any program may use these ports without special privileges. Table 2-1 shows the well-known ports for the protocols that are discussed in this book. These assignments are not absolutely guaranteed; in particular, web servers often run on ports other than 80, either because multiple servers need to run on the same machine or because the person who installed the server doesn't have the root privileges needed to run it on port 80. On Unix systems, a fairly complete listing of assigned ports is stored in the file */etc/services*.

Table 2-1. Well-known port assignments

Protocol	Port	Protocol	Purpose
echo	7	TCP/UDP	Echo is a test protocol used to verify that two machines are able to connect by having one echo back the other's input.
discard	9	TCP/UDP	Discard is a less useful test protocol in which all data received by the server is ignored.
daytime	13	TCP/UDP	Provides an ASCII representation of the current time on the server.
FTP data	20	TCP	FTP uses two well-known ports. This port is used to transfer files.
FTP	21	TCP	This port is used to send FTP commands like `put` and `get`.
SSH	22	TCP	Used for encrypted, remote logins.
telnet	23	TCP	Used for interactive, remote command-line sessions.
smtp	25	TCP	The Simple Mail Transfer Protocol is used to send email between machines.

Protocol	Port	Protocol	Purpose
time	37	TCP/UDP	A time server returns the number of seconds that have elapsed on the server since midnight, January 1, 1900, as a four-byte, signed, big-endian integer.
whois	43	TCP	A simple directory service for Internet network administrators.
finger	79	TCP	A service that returns information about a user or users on the local system.
HTTP	80	TCP	The underlying protocol of the World Wide Web.
POP3	110	TCP	Post Office Protocol Version 3 is a protocol for the transfer of accumulated email from the host to sporadically connected clients.
NNTP	119	TCP	Usenet news transfer; more formally known as the "Network News Transfer Protocol".
IMAP	143	TCP	Internet Message Access Protocol is a protocol for accessing mailboxes stored on a server.
RMI Registry	1099	TCP	The registry service for Java remote objects. This will be discussed in Chapter 18.

2.4. The Internet

The *Internet* is the world's largest IP-based network. It is an amorphous group of computers in many different countries on all seven continents (Antarctica included) that talk to each other using the IP protocol. Each computer on the Internet has at least one unique IP address by which it can be identified. Most of them also have at least one name that maps to that IP address. The Internet is not owned by anyone, although pieces of it are. It is not governed by anyone, which is not to say that some governments don't try. It is simply a very large collection of computers that have agreed to talk to each other in a standard way.

The Internet is not the only IP-based network, but it is the largest one. Other IP networks are called *internets* with a little *i*: for example, a corporate IP network that is not connected to the

Internet. *Intranet* is a current buzzword that loosely describes corporate practices of putting lots of data on internal web servers.

Unless you're working in a high security environment that's physically disconnected from the broader network, it's likely that the internet you'll be using is the Internet. To make sure that hosts on different networks on the Internet can communicate with each other, a few rules need to be followed that don't apply to purely internal internets. The most important rules deal with the assignment of addresses to different organizations, companies, and individuals. If everyone picked the Internet addresses they wanted at random, conflicts would arise almost immediately when different computers showed up on the Internet with the same address.

2.4.1. Internet Address Classes

To avoid this problem, blocks of IPv4 addresses are assigned to Internet Service Providers (ISPs) by their regional Internet registry. When a company or an organization wants to set up an IP-based network connected to the Internet, their ISP gives them a block of addresses. Traditionally, these blocks come in three sizes called Class A, Class B, and Class C. A Class C address block specifies the first three bytes of the address; for example, 199.1.32. This allows room for 254 individual addresses from 199.1.32.1 to 199.1.32.254.[1] A class B address block only specifies the first two bytes of the addresses an organization may use; for instance, 167.1. Thus, a class B address has room for 65,024 different hosts (256 Class C size blocks times 254 hosts per Class C block). A class A address block only specifies the first byte of the address range—for instance, 18—and therefore has room for over 16 million nodes.

[1] Addresses with the last byte either .0 or .255 are reserved and should never actually be assigned to hosts.

There are also Class D and E addresses. Class D addresses are used for IP multicast groups, and will be discussed at length in Chapter 14. Class D addresses all begin with the four bits 1110. Class E addresses begin with the five bits 11110 and are reserved for future extensions to the Internet.

There's no block with a size between a class A and a Class B, or Class B and a Class C. This has become a problem because there are many organizations with more than 254 computers connected to the Internet but less than 65,024. If each of these organizations gets a full Class B block, many addresses are wasted. There's a limited number of IPv4 addresses—about 4.2 billion, to be precise. That sounds like a lot, but it gets crowded quickly when you can easily waste fifty or sixty thousand addresses at a shot.

There are also many networks, such as the author's own personal basement-area network, that have a few to a few dozen computers but not 255. To more efficiently allocate the limited address space, Classless Inter-Domain Routing (CIDR) was invented. CIDR mostly (though not completely) replaces the whole A, B, C, D, E addressing scheme with one based on a specified numbers of prefix bits. These prefixes are generally written as */nn*, where *nn* is a two-digit number specifying the number of bits in the network portion of the address. The number after the / indicates the number of fixed prefix bits. Thus, a /24 fixes the first 24 bits in the address, leaving 8 bits available to distinguish individual nodes. This allows 256 nodes, and is equivalent to an old style Class C. A /19 fixes 19 bits, leaving 13 for individual nodes within the network. It's equivalent to 32 separate Class C networks or an eighth of a Class B. A /28, generally the smallest you're likely to encounter in practice, leaves only four bits for identifying local nodes. It can handle networks with up to 16 nodes. CIDR also carefully specifies which address blocks are associated with which ISPs. This scheme helps keep Internet routing tables smaller and more manageable than they would be under the old system.

Several address blocks and patterns are special. All IPv4 addresses that begin with 10., 172.16. through 172.31., and 192.168. are deliberately unassigned. They can be used on internal networks, but no host using addresses in these blocks is allowed onto the global Internet. These *non-routable* addresses are useful for building private networks that can't be seen from the rest of the Internet or for building a large network when you've only been assigned a class C address block. IPv4 addresses beginning with 127 (most commonly 127.0.0.1) always mean the *local loopback address*. That is, these addresses always point to the local computer, no matter which computer you're running on. The hostname for this address is generally *localhost*. In IPv6 0:0:0:0:0:0:0:1 (a.k.a. ::1) is the loopback address. The address 0.0.0.0 always refers to the originating host, but may only be used as a source address, not a destination. Similarly, any IPv4 address that begins with 0.0 is assumed to refer to a host on the same local network.

2.4.2. Network Address Translation

For reasons of both security and address space conservation, many smaller networks, such as the author's home network, use *network address translation* (NAT). Rather than allotting even a /28, my ISP gives me a single address, 216.254.85.72. Obviously, that won't work for the dozen or so different computers and other devices running in my apartment at any one time. Instead, I assign each one of them a different address in the non-routable block *192.168.254.xxx*. When they connect to the internet, they have to pass through a router my ISP sold me that translates the internal addresses into the external addresses.

The router watches my outgoing and incoming connections and adjusts the addresses in the IP packets. For an outgoing packet, it changes the source address to the router's external address (216.254.85.72 on my network). For an incoming packet, it changes the destination address to one of the local addresses, such as 192.168.254.12. Exactly how it keeps track of which connections come from and are aimed at which internal computers is not particularly important to a Java programmer. As long as your machines are configured properly, this process is mostly transparent to Java programs. You just need to remember that the external and internal addresses may not be the same. From outside my network, nobody can talk to my system at 192.168.254.12 unless I initiate the connection, or unless I configure my router to forward requests addressed to 216.254.85.72 to 192.168.254.12. If the router is safe, then the rest of the network is too. On the other hand, if someone does crack the router or one of the servers behind the router that is mapped to 216.254.85.72, I'm hosed. This is why I installed a firewall as the next line of defense.

2.4.3. Firewalls

There are some naughty people on the Internet. To keep them out, it's often helpful to set up one point of access to a local network and check all traffic into or out of that access point. The hardware and software that sit between the Internet and the local network, checking all the data that comes in or out to make sure it's kosher, is called a *firewall*. The firewall is often part of the router that connects the local network to the broader Internet and may perform other tasks, such as network address translation. Then again, the firewall may be a separate machine. Modern operating systems like Mac OS X and Red Hat Linux often have built-in personal firewalls that monitor just the traffic sent to that one machine. Either way, the firewall is responsible for inspecting each packet that passes into or out of its network interface and accepting it or rejecting it according to a set of rules.

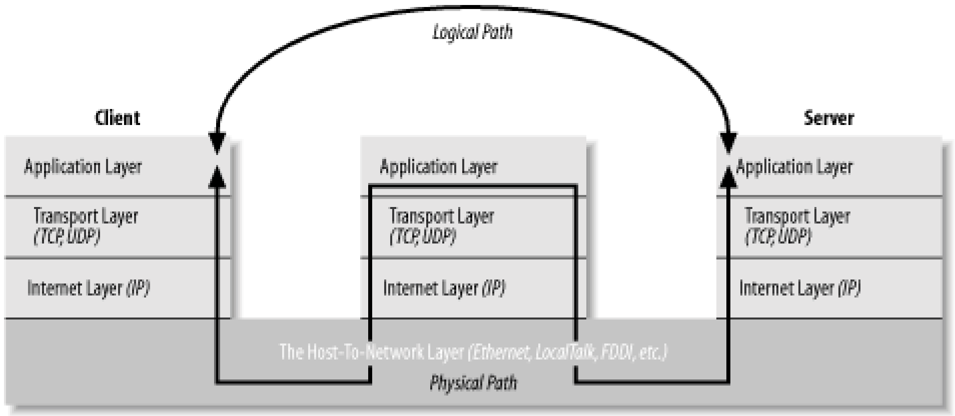
Filtering is usually based on network addresses and ports. For example, all traffic coming from the Class C network 193.28.25 may be rejected because you had bad experiences with hackers from that network in the past. Outgoing Telnet connections may be allowed, but incoming Telnet connections may not. Incoming connections on port 80 (web) may be allowed, but only to the corporate web server. More intelligent firewalls look at the contents of the packets to determine whether to accept or reject them. The exact configuration of a firewall—which packets of data are and are not allowed to pass through—depends on the security needs of an individual site. Java doesn't have much to do with firewalls—except in so far as they often get in your way.

2.4.4. Proxy Servers

Proxy servers are related to firewalls. If a firewall prevents hosts on a network from making direct connections to the outside world, a proxy server can act as a go-between. Thus, a machine that is prevented from connecting to the external network by a firewall would make a request for a web page from the local proxy server instead of requesting the web page directly from the remote web server. The proxy server would then request the page from the web server and forward the response back to the original requester. Proxies can also be used for FTP services and other connections. One of the security advantages of using a proxy server is that external hosts only find out about the proxy server. They do not learn the names and IP addresses of the internal machines, making it more difficult to hack into internal systems.

While firewalls generally operate at the level of the transport or internet layer, proxy servers normally operate at the application layer. A proxy server has a detailed understanding of some application level protocols, such as HTTP and FTP. (The notable exception are SOCKS proxy servers that operate at the transport layer, and can proxy for all TCP and UDP connections regardless of application layer protocol.) Packets that pass through the proxy server can be examined to ensure that they contain data appropriate for their type. For instance, FTP packets that seem to contain Telnet data can be rejected. Figure 2-3 shows how proxy servers fit into the layer model.

Figure 2-3. Layered connections through a proxy server



As long as all access to the Internet is forwarded through the proxy server, access can be tightly controlled. For instance, a company might choose to block access to www.playboy.com but allow access to www.microsoft.com. Some companies allow incoming FTP but disallow outgoing FTP so confidential data cannot be as easily smuggled out of the company. Other companies have begun using proxy servers to track their employees' web usage so they can see who's using the Internet to get tech support and who's using it to check out the Playmate of the Month. Such monitoring of employee behavior is controversial and not exactly an indicator of enlightened management techniques.

Proxy servers can also be used to implement local caching. When a file is requested from a web server, the proxy server first checks to see if the file is in its cache. If the file is in the cache, the proxy serves the file from the cache rather than from the Internet. If the file is not in the cache, the proxy server retrieves the file, forwards it to the requester, and stores it in the cache for the next time it is requested. This scheme can significantly reduce load on an Internet connection and greatly improve response time. America Online runs one of the largest farm of proxy servers in the world to speed the transfer of data to its users. If you look at a web server logfile, you'll probably find some hits from clients in the aol.com domain, but not as many as you'd expect given the more than twenty million AOL subscribers. That's because AOL proxy servers supply many pages out of their cache rather than re-requesting them for each user. Many other large ISPs do similarly.

The biggest problem with proxy servers is their inability to cope with all but a few protocols. Generally established protocols like HTTP, FTP, and SMTP are allowed to pass through, while newer protocols like Gnutella are not. (Some network administrators would consider this a feature.) In the rapidly changing world of the Internet, this is a significant disadvantage. It's a particular disadvantage for Java programmers because it limits the effectiveness of custom protocols. In Java, it's easy and often useful to create a new protocol that is optimized for your application. However, no proxy server will ever understand these one-of-a-kind protocols. Consequently, some developers have taken to tunneling their protocols through HTTP, most notably with SOAP. However, this has a significant negative impact on security. The firewall is normally there for a reason, not just to annoy Java programmers.

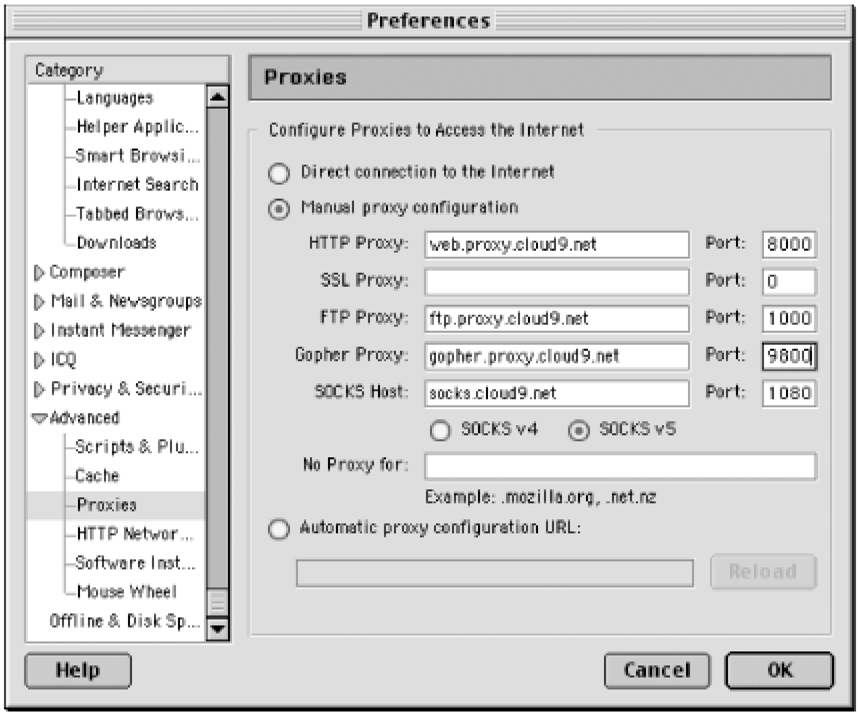
Applets that run in web browsers use the proxy server settings of the web browser itself, generally set in a dialog box (possibly hidden several levels deep in the preferences) like the one in Figure 2-4. Standalone Java applications can indicate the proxy server to use by setting the `socksProxyHost` and `socksProxyPort` properties (if you're using a SOCKS proxy server), or `http.proxySet`, `http.proxyHost`, `http.proxyPort`, `https.proxySet`, `https.proxyHost`, `https.proxyPort`, `ftpProxySet`, `ftpProxyHost`, `ftpProxyPort`, `gopherProxySet`, `gopherProxyHost`, and `gopherProxyPort` system properties (if you're using protocol-specific proxies). You can set system properties from the command line using the `-D` flag, like this:

```
java -DsocksProxyHost=
                socks.cloud9.net
             -DsocksProxyPort=
                1080
              MyClass
```

You can use any other convenient means to set these system properties, such as including them in the *appletviewer.properties* file, like this:

```
ftpProxySet=true
ftpProxyHost=ftp.proxy.cloud9.net
ftpProxyPort=1000
gopherProxySet=true
gopherProxyHost=gopher.proxy.cloud9.net
gopherProxyPort=9800
http.proxySet=true
http.proxyHost=web.proxy.cloud9.net
http.proxyPort=8000
https.proxySet=true
https.proxyHost=web.proxy.cloud9.net
https.proxyPort=8001
```

Figure 2-4. Netscape Navigator proxy server settings



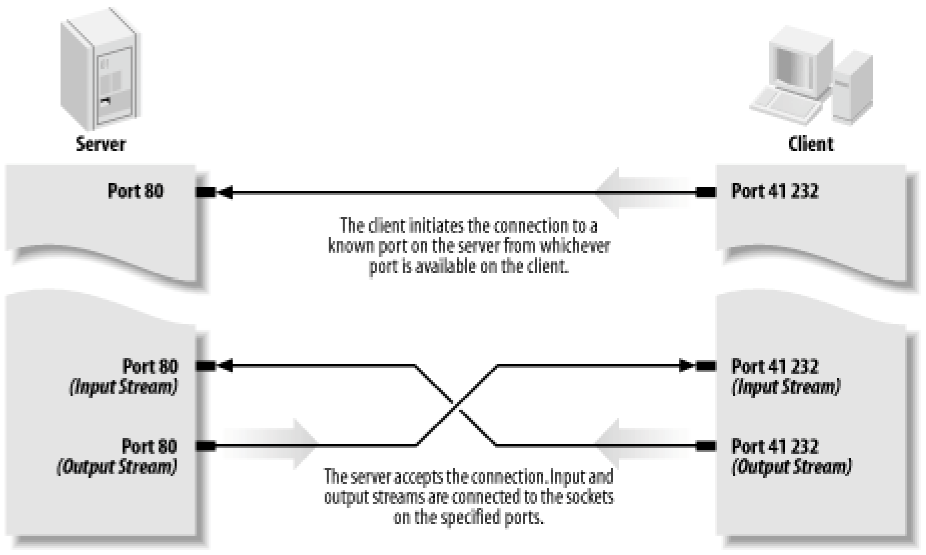
Direct connection to the Internet
[✓ Manual proxy configuration]

HTTP Proxy: web.proxy.cloud9.net Port: 8000
SSL Proxy: Port: 0
FTP Proxy: ftp.proxy.cloud9.net Port: 1000
Gopher Proxy: gopher.proxy.cloud9.net Port: 9800
SOCKS Host: socks.cloud9.net Port: 1080
SOCKS v4 [✓ SOCKS v5]
No Proxy for:
Example: .mozilla.org, .net.nz
Automatic proxy configuration URL:

2.5. The Client/Server Model

Most modern network programming is based on a client/server model. A client/server application typically stores large quantities of data on an expensive, high-powered server while most of the program logic and the user-interface is handled by client software running on relatively cheap personal computers. In most cases, a server primarily sends data while a client primarily receives it, but it is rare for one program to send or receive exclusively. A more reliable distinction is that a client initiates a conversation while a server waits for clients to start conversations with it. Figure 2-5 illustrates both possibilities. In some cases, the same program may be both a client and a server.

Figure 2-5. A client/server connection


Server
Client
Port 80
Port 41 232
The client initiates the connection to a known port on the server from whichever port is available on the client.
Port 80 (Input Stream)
Port 41 232 (Input Stream)
Port 80 (Output Stream)
Port 41 232 (Output Stream)
The server accepts the connection. Input and output streams are connected to the sockets on the specified ports.

Some servers process and analyze the data before sending the results to the client. Such servers are often referred to as "application servers" to distinguish them from the more common file servers and database servers. A file or database server will retrieve information and send it to a client, but it won't process that information. In contrast, an application server might look at an order entry database and give the clients reports about monthly sales trends. An application server is not a server that serves files that happen to be applications.

You are already familiar with many examples of client/server systems. In 2004, the most popular client/server system on the Internet is the Web. Web servers like Apache respond to requests from web clients like Firefox. Data is stored on the web server and is sent out to the clients that request it. Aside from the initial request for a page, almost all data is transferred from the server to the client, not from the client to the server. Web servers that use CGI programs double as application and file servers. FTP is an older service that fits the client/server model. FTP uses different application protocols and different software, but is still split into FTP servers that send files and FTP clients that receive files. People often use FTP to upload files from the client to the server, so it's harder to say that the data transfer is primarily in one direction, but it is still true that an FTP client initiates the connection and the FTP server responds.

Not all applications fit easily into a client/server model. For instance, in networked games, it seems likely that both players will send data back and forth roughly equally (at least in a fair game). These sorts of connections are called *peer-to-peer*. The telephone system is the classic example of a peer-to-peer network. Each phone can either call another phone or be called by another phone. You don't have to buy one phone to send calls and another to receive them.

Java does not have explicit peer-to-peer communication in its core networking API (though Sun has implemented it in a separate open source project called JXTA). However, applications can easily offer peer-to-peer communications in several ways, most commonly by acting as both a server and a client. Alternately, the peers can communicate with each other through an intermediate server program that forwards data from one peer to the other peers. This is especially useful for applets with a security manager that restricts them from talking directly to each other.

2.6. Internet Standards

This book discusses several application-layer Internet protocols, most notably HTTP. However, this is not a book about those protocols and it tries not to say more than the minimum you need to know. If you need detailed information about any protocol, the definitive source is the standards document for the protocol.

While there are many standards organizations in the world, the two that produce most of the standards relevant to network programming and protocols are the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C). The IETF is a relatively informal, democratic body open to participation by any interested party. Its standards are based on "rough consensus and running code" and tend to follow rather than lead implementations. IETF standards include TCP/IP, MIME, and SMTP. The W3C, by contrast, is a vendor organization, controlled by dues-paying member corporations, that explicitly excludes participation by individuals. For the most part, the W3C tries to define standards in advance of implementation. W3C standards include HTTP, HTML, and XML.

2.6.1. IETF RFCs

IETF standards and near-standards are published as Internet drafts and requests for comments (RFCs). RFCs and Internet drafts range from informational documents of general interest to detailed specifications of standard Internet protocols like FTP. RFCs that document a standard or a proposed standard are published only with the approval of the Internet Engineering Steering Group (IESG) of the IETF. All IETF approved standards are RFCs, but not all RFCs are IETF standards. RFCs are available from many locations on the Internet, including http://www.faqs.org/rfc/ and http://www.ietf.org/rfc.html. For the most part RFCs, particularly standards-oriented RFCs, are very technical, turgid, and nearly incomprehensible. Nonetheless, they are often the only complete and reliable source of information about a particular protocol.

Most proposals for a standard begin when a person or group gets an idea and builds a prototype. The prototype is incredibly important. Before something can become an IETF standard, it must actually exist and work. This requirement ensures that IETF standards are at least feasible, unlike the standards promulgated by some other organizations. If the prototype becomes popular outside its original developers and if other organizations begin implementing their own versions of the protocol, a *working group* may be formed under the auspices of the IETF. This working group attempts to document the protocol in an *Internet-Draft*. Internet-Drafts are working documents and change frequently to reflect experience with the protocol. The experimental implementations and the Internet-Draft evolve in rough synchronization, until eventually the working group agrees that the protocol is ready to become a formal standard. At this point, the proposed specification is submitted to the IESG.

The proposal goes through six states or maturity levels as it follows the standardization track:

- Experimental
- Proposed standard
- Draft standard
- Standard
- Informational
- Historic

For some time after the proposal is submitted, it is considered *experimental*. The experimental stage does not imply that the protocol is not solid or that it is not widely used; unfortunately, the standards process usually lags behind *de facto* acceptance of the standard. If the IESG likes the experimental standard or it is in widespread use, the IESG will assign it an RFC number and publish it as an experimental RFC, generally after various changes.

If the experimental standard holds up well in further real world testing, the IESG may advance it to the status of *proposed standard*. A proposed standard is fairly loose, and is based on the experimental work of possibly as little as one organization. Changes may still be made to a protocol in this stage.

Once the bugs appear to have been worked out of a proposed standard and there are at least two independent implementations, the IESG may recommend that a proposed standard be promoted to a *draft standard*. A draft standard will probably not change too much before eventual standardization unless major flaws are found. The primary purpose of a draft standard is to clean up the RFC that documents the protocol and make sure the documentation conforms to actual practice, rather than to change the standard itself.

When a protocol completes this, it becomes an official Internet *standard*. It is assigned an STD number and is published as an STD in addition to an RFC. The absolute minimum time for a standard to be approved as such is 10 months, but in practice, the process almost always takes much longer. The commercial success of the Internet hasn't helped, since standards must now be worked out in the presence of marketers, vulture capitalists, lawyers, NSA spooks, and others with vested interests in seeing particular technologies succeed or fail. Therefore, many of the "standards" that this book references are in either the experimental, proposed, or draft stage. As of publication, there are over 3,800 RFCs. Less than one hundred of these have become STDs, and some of those that have are now obsolete. RFCs relevant to this book are detailed in Table 2-2.

Some RFCs that do not become standards are considered *informational*,. These include RFCs that specify protocols that are widely used but weren't developed within the normal Internet standards track, and haven't been through the formal standardization process. For example, NFS, originally developed by Sun, is described in the informational RFC 1813. Other informational RFCs provide useful information (like users' guides), but don't document a protocol. For example, RFC 1635, *How to Use Anonymous FTP*, is an informational RFC.

Finally, changing technology and increasing experience renders some protocols and their associated RFCs obsolete. These are classified as *historic*. Historic protocols include IMAP3 (replaced by IMAP4), POP2 (replaced by POP3), and Remote Procedure Call Version 1 (replaced by Remote Procedure Call Version 2).

In addition to its maturity level, a protocol has a requirement level. The possible requirement levels are:

Not recommended
: Should not be implemented by anyone.

Limited use
: May have to be implemented in certain unusual situations but won't be needed by most hosts. Mainly these are experimental protocols.

Elective
: Can be implemented by anyone who wants to use the protocol. For example, RFC 2045, *Multipurpose Internet Mail Extensions*, is a Draft Elective Standard.

Recommended
: Should be implemented by Internet hosts that don't have a specific reason not to implement it. Most protocols that you are familiar with (like TCP and UDP, SMTP for email, Telnet for remote login, etc.) are recommended.

Required
: Must be implemented by all Internet hosts. There are very few required protocols. IP itself is one (RFC 791), but even protocols as important as TCP or UDP are only recommended. A standard is only required if it is absolutely essential to the functioning of a host on the Internet.

Table 2-2 lists the RFCs and STDs that provide formal documentation for the protocols discussed in this book.

Table 2-2. Selected Internet RFCs

RFC	Title	Maturity level	Requirement level	Description
RFC 3300 STD 1	Internet Official Protocol Standards	Standard	Required	Describes the standardization process and the current status of the different Internet protocols.
RFC 1122 RFC 1123 STD 3	Host Requirements	Standard	Required	Documents the protocols that must be supported by all Internet hosts at different layers (data link layer, IP layer, transport layer, and application layer).
RFC 791 RFC 919 RFC 922 RFC 950 STD 5	Internet Protocol	Standard	Required	The IP internet layer protocol.
RFC 768 STD 6	User Datagram Protocol	Standard	Recommended	An unreliable, connectionless transport layer protocol.
RFC 792 STD 5	Internet Control Message Protocol (ICMP)	Standard	Required	An internet layer protocol that uses raw IP datagrams but is not supported by Java. Its most familiar use is the *ping* program.
RFC 793 STD 7	Transmission Control Protocol	Standard	Recommended	A reliable, connection-oriented, streaming transport layer protocol.

RFC	Title	Maturity level	Requirement level	Description
RFC 2821	Simple Mail Transfer Protocol	Proposed standard	Recommended	The application layer protocol by which one host transfers email to another host. This standard doesn't say anything about email user interfaces; it covers the mechanism for passing email from one computer to another.
RFC 822 STD 11	Format of Electronic Mail Messages	Standard	Recommended	The basic syntax for ASCII text email messages. MIME is designed to extend this to support binary data while ensuring that the messages transferred still conform to this standard.
RFC 854 RFC 855 STD 8	Telnet Protocol	Standard	Recommended	An application-layer remote login service for command-line environments based around an abstract network virtual terminal (NVT) and TCP.
RFC 862 STD 20	Echo Protocol	Standard	Recommended	An application-layer protocol that echoes back all data it receives over both TCP and UDP; useful as a debugging tool.
RFC 863 STD 21	Discard Protocol	Standard	Elective	An application layer protocol that receives packets of data over both TCP and UDP and sends no response to the client; useful as a debugging tool.
RFC 864 STD 22	Character Generator Protocol	Standard	Elective	An application layer protocol that sends an indefinite sequence of ASCII characters to any client that connects over either TCP or UDP; also useful as a debugging tool.

RFC	Title	Maturity level	Requirement level	Description
RFC 865 STD 23	Quote of the Day	Standard	Elective	An application layer protocol that returns a quotation to any user who connects over either TCP or UDP and then closes the connection.
RFC 867 STD 25	Daytime Protocol	Standard	Elective	An application layer protocol that sends a human-readable ASCII string indicating the current date and time at the server to any client that connects over TCP or UDP. This contrasts with the various NTP and Time Server protocols, which do not return data that can be easily read by humans.
RFC 868 STD 26	Time Protocol	Standard	Elective	An application layer protocol that sends the time in seconds since midnight, January 1, 1900 to a client connecting over TCP or UDP. The time is sent as a machine-readable, 32-bit signed integer. The standard is incomplete in that it does not specify how the integer is encoded in 32 bits, but in practice a two's complement, big-endian integer is used.
RFC 959 STD 9	File Transfer Protocol	Standard	Recommended	An optionally authenticated, two-socket application layer protocol for file transfer that uses TCP.
RFC 977	Network News Transfer Protocol	Proposed standard	Elective	The application layer protocol by which Usenet news is transferred from machine to machine over TCP; used by both news clients talking to

RFC	Title	Maturity level	Requirement level	Description
				news servers and news servers talking to each other.
RFC 1034 RFC 1035 STD 13	Domain Name System	Standard	Recommended	The collection of distributed software by which hostnames that human beings can remember, like www.oreilly.com, are translated into numbers that computers can understand, like 198.112.208.11. This STD defines how domain name servers on different hosts communicate with each other using UDP.
RFC 1112	Host Extensions for IP Multicasting	Standard	Recommended	The internet layer methods by which conforming systems can direct a single packet of data to multiple hosts. This is called multicasting; Java's support for multicasting is discussed in Chapter 14.
RFC 1153	Digest Message Format for Mail	Experimental	Limited use	A format for combining multiple postings to a mailing list into a single message.
RFC 1288	Finger Protocol	Draft standard	Elective	An application layer protocol for requesting information about a user at a remote site. It can be a security risk.
RFC 1305	Network Time Protocol (Version 3)	Draft standard	Elective	A more precise application layer protocol for synchronizing clocks between systems that attempts to account for network latency.
RFC 1738	Uniform Resource Locators	Proposed standard	Elective	Full URLs like http://www.amnesty.org/ and *ftp://*

RFC	Title	Maturity level	Requirement level	Description
				ftp.ibiblio.org/pub/multimedia/chinese-music/Dream_Of_Red_Mansion/HLM04 .Handkerchief.au.
RFC 1808	Relative Uniform Resource Locators	Proposed standard	Elective	Partial URLs like */javafaq/books/* and *../examples/07/index.html* used as values of the `HREF` attribute of an HTML A element.
RFC 1939 STD 53	Post Office Protocol, Version 3	Standard	Elective	An application-layer protocol used by sporadically connected email clients such as Eudora to retrieve mail from a server over TCP.
RFC 1945	Hypertext Transfer Protocol (HTTP 1.0)	Informational	N/A	Version 1.0 of the application layer protocol used by web browsers talking to web servers over TCP; developed by the W3C rather than the IETF.
RFC 2045 RFC 2046 RFC 2047	Multipurpose Internet Mail Extensions	Draft standard	Elective	A means of encoding binary data and non-ASCII text for transmission through Internet email and other ASCII-oriented protocols.
RFC 2068	Hypertext Transfer Protocol (HTTP 1.1)	Proposed standard	Elective	Version 1.1 of the application layer protocol used by web browsers talking to web servers over TCP.
RFC 2141	Uniform Resource Names (URN) Syntax	Proposed standard	Elective	Similar to URLs but intended to refer to actual resources in a persistent fashion rather than the transient location of those resources.

RFC	Title	Maturity level	Requirement level	Description
RFC 2373	IP Version 6 Addressing Architecture	Proposed standard	Elective	The format and meaning of IPv6 addresses.
RFC 2396	Uniform Resource Identifiers (URI): Generic Syntax	Proposed standard	Elective	Similar to URLs but cut a broader path. For instance, ISBN numbers may be URIs even if the book cannot be retrieved over the Internet.
RFC 3501	Internet Message Access Protocol Version 4rev1	Proposed standard	Elective	A protocol for remotely accessing a mailbox stored on a server including downloading messages, deleting messages, and moving messages into and out of different folders.

The IETF has traditionally worked behind the scenes, codifying and standardizing existing practice. Although its activities are completely open to the public, it's traditionally been very low-profile. There simply aren't that many people who get excited about the details of network arcana like the Internet Gateway Message Protocol (IGMP). The participants in the process have mostly been engineers and computer scientists, including many from academia as well as the corporate world. Consequently, despite often vociferous debates about ideal implementations, most serious IETF efforts have produced reasonable standards.

Unfortunately, that can't be said of the IETF's efforts to produce web (as opposed to Internet) standards. In particular, the IETF's early effort to standardize HTML was a colossal failure. The refusal of Netscape and other key vendors to participate or even acknowledge the process was a crucial problem. That HTML was simple enough and high-profile enough to attract the attention of assorted market-droids and random flamers didn't help matters either. Thus, in October 1994 the World Wide Web Consortium was formed as a vendor-controlled body that might be able to avoid the pitfalls that plagued the IETF's efforts to standardize HTML and HTTP.

2.6.2. W3C Recommendations

Although the W3C standardization process is similar to the IETF process (a series of working drafts hashed out on mailing lists resulting in an eventual specification), the W3C is a fundamentally different organization. Whereas the IETF is open to participation by anyone, only corporations and other organizations may become members of the W3C. Individuals are specifically excluded. Furthermore, although nonprofit organizations like the World Wide Web Artists Consortium (WWWAC) may join the W3C, only the employees of these organizations may participate in W3C activities. Their volunteer members are not welcome. Specific individual experts are occasionally invited to participate on a particular working group even though they are not employees of a W3C member company. However, the number of such individuals is quite small relative to the number of interested experts in the broader community. Membership in the W3C costs $50,000 a year ($5,000 a year for nonprofits) with a minimum 3-year commitment. Membership in the IETF costs $0 a year with no commitment beyond a willingness to participate. And although many people participate in developing W3C standards, each standard is ultimately approved or vetoed by one individual, W3C director Tim Berners-Lee. IETF standards are approved by a consensus of the people who worked on the standard. Clearly, the IETF is a much more democratic (some would say anarchic) and open organization than the W3C.

Despite the W3C's strong bias toward the corporate members that pay its bills, it has so far managed to do a better job of navigating the politically tricky waters of Web standardization than the IETF. It has produced several HTML standards, as well as a variety of others such as HTTP, PICS, XML, CSS, MathML, and more. The W3C has had considerably less success in convincing vendors like Netscape and Microsoft to fully and consistently implement its standards.

The W3C has five basic levels of standards:

Note

A note is generally one of two things: either an unsolicited submission by a W3C member (similar to an IETF Internet draft) or random musings by W3C staff or related parties that do not actually describe a full proposal (similar to an IETF informational RFC). Notes will not necessarily lead to the formation of a working group or a W3C recommendation.

Working drafts

A working draft is a reflection of the current thinking of some (not necessarily all) members of a working group. It should eventually lead to a proposed recommendation, but by the time it does so it may have changed substantially.

Candidate recommendation

A candidate recommendation indicates that the working draft has reached consensus on all major issues and is ready for third-party comment and implementations. If the implementations do not uncover any obstructions, the spec can be promoted to a proposed recommendation.

Proposed recommendation

A proposed recommendation is mostly complete and unlikely to undergo more than minor editorial changes. The main purpose of a proposed recommendation is to work out bugs in the specification document rather than in the underlying technology being documented.

Recommendation

A recommendation is the highest level of W3C standard. However, the W3C is very careful not to actually call this a "standard" for fear of running afoul of antitrust statutes. The W3C describes a recommendation as a "work that represents consensus within W3C and has the Director's stamp of approval. W3C considers that the ideas or technology specified by a Recommendation are appropriate for widespread deployment and promote W3C's mission."

The W3C has not been around long enough to develop a need for a historical or informational standard status. Another difference the IETF and the W3C is that the W3C process rarely fails to elevate a standard to full recommendation status once work has actively commenced—that is, once a working group has been formed. This contrasts markedly with the IETF, which has more than a thousand proposed and draft standards, but only a few dozen actual standards.

PR Standards

In recent years, companies seeking a little free press or perhaps a temporary boost to their stock price have abused both the W3C and IETF standards processes. The IETF will accept a submission from anyone, and the W3C will accept a submission from any W3C member. The IETF calls these submissions "Internet drafts" and publishes them for six months before deleting them. The W3C refers to such submissions as "acknowledged submissions" and publishes them indefinitely. However, neither organization actually promises to do more than acknowledge receipt of these documents. In particular, they do not promise to form a working group or begin the standardization process. Nonetheless, press releases invariably misrepresent the submission of such a document as a far more significant event than it actually is. PR reps can generally count on suckering at least a few clueless reporters who aren't up to speed on the intimate details of the standardization process. However, you should recognize these ploys for what they are.