1.1 Networks, Packets, and Protocols

A computer network consists of machines interconnected by communication channels. We call  
these machines *hosts* and *routers*.

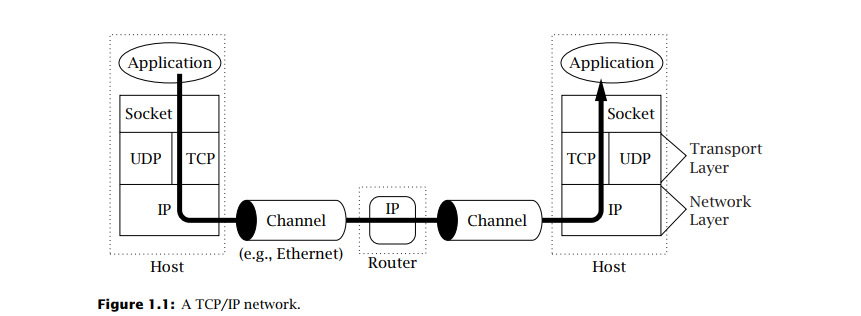
Hosts are computers that run applications such as your Web browser, your IM agent, or a file-sharing program. The application programs running on hosts  
are the real “users” of the network.

Routers are machines whose job is to relay, or *forward*, information from one communication channel to another.

They may run programs but typically do not run application programs.

Routers are important simply because it is not practical to connect every host directly  
to every other host. Instead, a few hosts connect to a router, which connects to other routers,  
and so on to form the network. This arrangement lets each machine get by with a relatively  
small number of communication channels; most hosts need only one. Programs that exchange  
information over the network, however, do not interact directly with routers and generally  
remain blissfully unaware of their existence.

A *protocol* is an agreement about the packets exchanged by communicating programs  
and what they mean. A protocol tells how packets are structured. A protocol is usually designed to solve a specific problem using given capabilities. For example, the *HyperText Transfer Protocol (HTTP)* solves the problem of transferring hypertext objects between servers, where they are stored or generated, and Web  
browsers that make them visible and useful to users. Instant messaging protocols solve the  
problem of enabling two or more users to exchange brief text messages.



In TCP/IP, the bottom layer consists of the underlying communication channels

The single network layer protocol in the TCP/IP suite is the Internet Protocol; it  
solves the problem of making the sequence of channels and routers between any two hosts  
look like a single host-to-host channel

The Internet Protocol provides a *datagram* service: every packet is handled and delivered  
by the network independently, like letters or parcels sent via the postal system. To make this  
work, each IP packet has to contain the *address* of its destination, just as every package that  
you mail is addressed to somebody

Although most delivery companies guarantee delivery of a package, IP is only a best-effort protocol: it attempts to deliver each packet, but it can (and occasionally does) lose, reorder, or duplicate packets in transit through the network.

The layer above IP is called the *transport layer*. It offers a choice between two protocols:  
TCP and UDP. Each builds on the service provided by IP, but they do so in different ways to  
provide different kinds of transport, which are used by *application protocols* with different  
needs.

TCP and UDP have one function in common: addressing. Both TCP and UDP use addresses, called *port numbers*, to identify applications within hosts. TCP and UDP are called *end-to-end transport protocols* because they carry data all the way from one program to another (whereas IP only carries data from one host to another).

TCP is designed to detect and recover from the losses, duplications, and other errors that may occur in the host-to-host channel provided by IP. TCP provides a *reliable byte-stream* channel so that applications do not have to deal with these problems. It is a *connectionoriented* protocol: before using it to communicate, two programs must first establish a TCP connection, which involves completing an exchange of *handshake messages* between the TCP implementations on the two communicating computers. Using TCP is also similar in many ways to file input/output (I/O). In fact, a file that is written by one program and read by another is a reasonable model of communication over a TCP connection.

UDP, on the other hand, does not attempt to recover from errors experienced by IP; it simply extends the IP best-effort datagram service so that it works between application programs instead of between hosts. Thus, applications that use UDP must be prepared to deal with losses, reordering, and so on.

**1.2 About Addresses**

In TCP/IP, it takes two pieces of information to identify a particular program: an *Internet address*, used by IP, and a *port number*, the additional address interpreted by the transport protocol (TCP or UDP).

Internet addresses are binary numbers. They come in two flavors, corresponding to the  
two versions of the Internet Protocol that have been standardized. The most common type is  
version 4 (“IPv4,” [14]); the other is version 6 (“IPv6,” [7]), which is just beginning to be deployed.

IPv4 addresses are 32 bits long; because this is only enough to identify about 4 billion distinct  
destinations, they are not really big enough for today’s Internet

More than half of the total address space has already been allocated.) For that reason, IPv6 was introduced. IPv6 addresses are 128 bits long.

IPv4 addresses are conventionally written as a group of four decimal numbers separated by periods (e.g., 10.1.2.3); this is called the *dotted-quad* notation. The four numbers in a dotted-quad string represent the contents of the four bytes of the Internet address—thus, each is a number between 0 and 255.

The sixteen bytes of an IPv6 address, on the other hand, are represented as groups  
of hexadecimal digits, separated by colons (e.g., 2000:fdb8:0000:0000:0001:00ab:853c:39a1).  
Each group of digits represents two bytes of the address; leading zeros may be omitted, so  
the fifth and sixth groups in the foregoing example might be rendered as just :1:ab:.

Technically, each Internet address refers to the connection between a host and an underlying communication channel—in other words, a *network interface*. A host may have several  
interfaces; it is not uncommon, for example, for a host to have connections to both wired(Ethernet) and wireless (WiFi) networks. Because each such network connection belongs to a  
single host, an Internet address identifies a host as well as its connection to the network.  
However, the converse is not true, because a single host can have multiple interfaces, and each  
interface can have multiple addresses. (In fact, the same interface can have both IPv4 and IPv6  
addresses.)

The *port number* in TCP or UDP is always interpreted relative to an Internet address

Port numbers are 16-bit unsigned binary numbers, so each one is in  
the range 1 to 65,535. (0 is reserved.)

In each version of IP, certain special-purpose addresses are defined. One of these that  
is worth knowing is the *loopback address*, which is always assigned to a special *loopback  
interface*, a virtual device that simply echoes transmitted packets right back to the sender.  
The loopback interface is very useful for testing because packets sent to that address are  
immediately returned back to the destination. Moreover, it is present on every host, and can  
be used even when a computer has no other interfaces (i.e., is not connected to the network).  
The loopback address for IPv4 is 127.0.0.1;1 for IPv6 it is 0:0:0:0:0:0:0:1

Finally, another class consists of *multicast* addresses. Whereas regular IP (sometimes  
called “unicast”) addresses refer to a single destination, multicast addresses potentially refer  
to an arbitrary number of destinations. Multicasting is an advanced subject that we cover  
briefly in Chapter 4. In IPv4, multicast addresses in dotted-quad format have a first number in  
the range 224 to 239. In IPv6, multicast addresses start with FF.

**1.3 About Names**

You should understand that the use of names instead of addresses is a convenience feature that is independent of the basic service provided by TCP/IP—you can write and use TCP/IP applications  
without ever using a name. When you use a name to identify a communication endpoint, the  
system does some extra work to *resolve* the name into an address. This extra step is often  
worth it for a couple of reasons. First, names are obviously easier for humans to remember  
than dotted-quads (or, in the case of IPv6, strings of hexadecimal digits). Second, names provide a level of indirection, which insulates users from IP address changes.

The name-resolution service can access information from a wide variety of sources. Two  
of the primary sources are the *Domain Name System (DNS)* and local configuration databases.

**1.4 Clients and Servers**

The terms *client* and *server* refer to these roles: the client program initiates communication,  
while the server program waits passively for and then responds to clients that contact it

Together, the client and server compose the *application*. The terms *client* and *server* are  
descriptive of the typical situation in which the server makes a particular capability—for  
example, a database service—available to any client that is able to communicate with it.

Whether a program is acting as a client or server determines the general form of its  
use of the sockets API to establish communication with its *peer*. (The client is the peer of  
the server and vice versa.) Beyond that, the client-server distinction is important because  
the client needs to know the server’s address and port initially, but not vice versa. With the  
sockets API, the server can, if necessary, learn the client’s address information when it receives  
the initial communication from the client.

How does a client find out a server’s IP address and port number? Usually, the client  
knows the name of the server it wants

Finding a server’s port number is a different story. In principle, servers can use any port,  
but the client must be able to learn what it is. In the Internet, there is a convention of assigning  
well-known port numbers to certain applications. The Internet Assigned Number Authority  
(IANA) oversees this assignment

**1.5 What Is a Socket?**

A *socket* is an abstraction through which an application may send and receive data, in much  
the same way as an open file handle allows an application to read and write data to stable  
storage. A socket allows an application to plug in to the network and communicate with other  
applications that are plugged in to the same network. Information written to the socket by  
an application on one machine can be read by an application on a different machine and vice  
versa.  
Different types of sockets correspond to different underlying protocol suites and  
different stacks of protocols within a suite. This book deals only with the TCP/IP protocol  
suite. The main types of sockets in TCP/IP today are *stream sockets* and *datagram sockets*.  
Stream sockets use TCP as the end-to-end protocol (with IP underneath) and thus provide  
a reliable byte-stream service. A TCP/IP stream socket represents one end of a TCP connection. Datagram sockets use UDP (again, with IP underneath) and thus provide a best-effort  
datagram service that applications can use to send individual messages up to about 65,500  
bytes in length. Stream and datagram sockets are also supported by other protocol suites, but this book deals only with TCP stream sockets and UDP datagram sockets. A TCP/IP socket is  
uniquely identified by an Internet address, an end-to-end protocol (TCP or UDP), and a port  
number. As you proceed, you will encounter several ways for a socket to become bound to  
an address.

Figure 1.2 depicts the logical relationships among applications, socket abstractions,  
protocols, and port numbers within a single host. Note that a single socket abstraction can  
be referenced by multiple application programs. Each program that has a reference to a particular socket can communicate through that socket. Earlier we said that a port identifies an  
application on a host. Actually, a port identifies a socket on a host. From Figure 1.2, we see  
that multiple programs on a host can access the same socket. In practice, separate programs  
that access the same socket would usually belong to the same application (e.g., multiple copies  
of a Web server program), although in principle they could belong to different applications.

