

downstream than upstream traffic. But P2P video streaming and file distribution applications shift upstream traffic from servers to residential ISPs, thereby putting significant stress on the ISPs. Future P2P applications need to be designed so that they are friendly to ISPs [Xie 2008].

2. *Security*. Because of their highly distributed and open nature, P2P applications can be a challenge to secure [Doucer 2002; Yu 2006; Liang 2006; Naoumov 2006; Dhungel 2008; LeBlond 2011].
3. *Incentives*. The success of future P2P applications also depends on convincing users to volunteer bandwidth, storage, and computation resources to the applications, which is the challenge of incentive design [Feldman 2005; Piatek 2008; Aperijs 2008; Liu 2010].

## 2.1.2 Processes Communicating

Before building your network application, you also need a basic understanding of how the programs, running in multiple end systems, communicate with each other. In the jargon of operating systems, it is not actually programs but **processes** that communicate. A process can be thought of as a program that is running within an end system. When processes are running on the same end system, they can communicate with each other with interprocess communication, using rules that are governed by the end system's operating system. But in this book we are not particularly interested in how processes in the same host communicate, but instead in how processes running on *different* hosts (with potentially different operating systems) communicate.

Processes on two different end systems communicate with each other by exchanging **messages** across the computer network. A sending process creates and sends messages into the network; a receiving process receives these messages and possibly responds by sending messages back. Figure 2.1 illustrates that processes communicating with each other reside in the application layer of the five-layer protocol stack.

### Client and Server Processes

A network application consists of pairs of processes that send messages to each other over a network. For example, in the Web application a client browser process exchanges messages with a Web server process. In a P2P file-sharing system, a file is transferred from a process in one peer to a process in another peer. For each pair of communicating processes, we typically label one of the two processes as the **client** and the other process as the **server**. With the Web, a browser is a client process and a Web server is a server process. With P2P file sharing, the peer that is downloading the file is labeled as the client, and the peer that is uploading the file is labeled as the server.

You may have observed that in some applications, such as in P2P file sharing, a process can be both a client and a server. Indeed, a process in a P2P file-sharing system can both upload and download files. Nevertheless, in the context of any given

communication session between a pair of processes, we can still label one process as the client and the other process as the server. We define the client and server processes as follows:

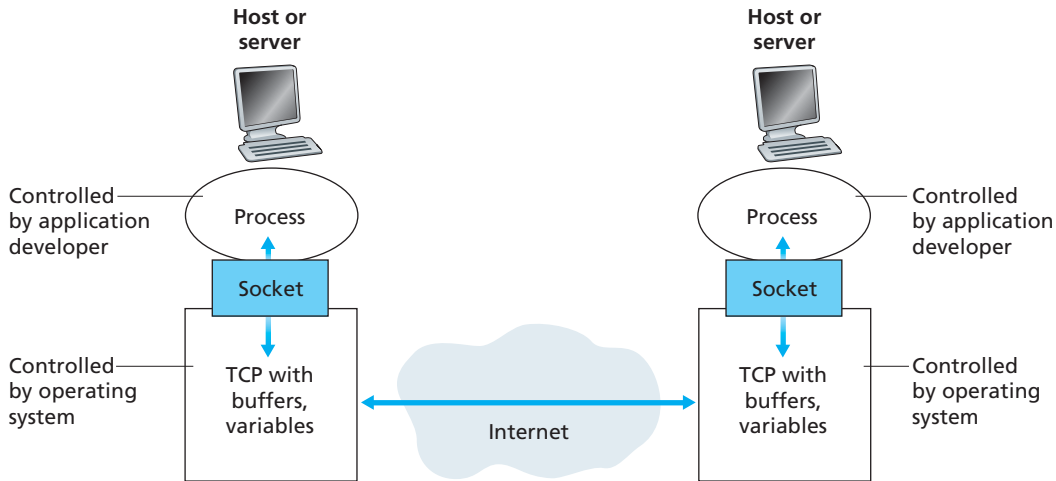
*In the context of a communication session between a pair of processes, the process that initiates the communication (that is, initially contacts the other process at the beginning of the session) is labeled as the **client**. The process that waits to be contacted to begin the session is the **server**.*

In the Web, a browser process initializes contact with a Web server process; hence the browser process is the client and the Web server process is the server. In P2P file sharing, when Peer A asks Peer B to send a specific file, Peer A is the client and Peer B is the server in the context of this specific communication session. When there's no confusion, we'll sometimes also use the terminology "client side and server side of an application." At the end of this chapter, we'll step through simple code for both the client and server sides of network applications.

## The Interface Between the Process and the Computer Network

As noted above, most applications consist of pairs of communicating processes, with the two processes in each pair sending messages to each other. Any message sent from one process to another must go through the underlying network. A process sends messages into, and receives messages from, the network through a software interface called a **socket**. Let's consider an analogy to help us understand processes and sockets. A process is analogous to a house and its socket is analogous to its door. When a process wants to send a message to another process on another host, it shoves the message out its door (socket). This sending process assumes that there is a transportation infrastructure on the other side of its door that will transport the message to the door of the destination process. Once the message arrives at the destination host, the message passes through the receiving process's door (socket), and the receiving process then acts on the message.

Figure 2.3 illustrates socket communication between two processes that communicate over the Internet. (Figure 2.3 assumes that the underlying transport protocol used by the processes is the Internet's TCP protocol.) As shown in this figure, a socket is the interface between the application layer and the transport layer within a host. It is also referred to as the **Application Programming Interface (API)** between the application and the network, since the socket is the programming interface with which network applications are built. The application developer has control of everything on the application-layer side of the socket but has little control of the transport-layer side of the socket. The only control that the application developer has on the transport-layer side is (1) the choice of transport protocol and (2) perhaps the ability to fix a few transport-layer parameters such as maximum buffer and maximum segment sizes (to be covered in Chapter 3). Once the application developer chooses a transport protocol (if a choice is available),



**Figure 2.3** ♦ Application processes, sockets, and underlying transport protocol

the application is built using the transport-layer services provided by that protocol. We'll explore sockets in some detail in Section 2.7.

### Addressing Processes

In order to send postal mail to a particular destination, the destination needs to have an address. Similarly, in order for a process running on one host to send packets to a process running on another host, the receiving process needs to have an address. To identify the receiving process, two pieces of information need to be specified: (1) the address of the host and (2) an identifier that specifies the receiving process in the destination host.

In the Internet, the host is identified by its **IP address**. We'll discuss IP addresses in great detail in Chapter 4. For now, all we need to know is that an IP address is a 32-bit quantity that we can think of as uniquely identifying the host. In addition to knowing the address of the host to which a message is destined, the sending process must also identify the receiving process (more specifically, the receiving socket) running in the host. This information is needed because in general a host could be running many network applications. A destination **port number** serves this purpose. Popular applications have been assigned specific port numbers. For example, a Web server is identified by port number 80. A mail server process (using the SMTP protocol) is identified by port number 25. A list of well-known port numbers for all Internet standard protocols can be found at <http://www.iana.org>. We'll examine port numbers in detail in Chapter 3.