1 Basic Concepts, Protocols and Terminology

Learning Objectives

After reading this chapter, you should:

- have a high level appreciation of the basic means by which messages are sent and received on modern networks;
- be familiar with the most important protocols used on networks;
- understand the addressing mechanism used on the Internet;
- understand the basic principles of client/server programming.

The fundamental purpose of this opening chapter is to introduce the underpinning network principles and associated terminology with which the reader will need to be familiar in order to make sense of the later chapters of this book. The material covered here is entirely generic (as far as any programming language is concerned) and it is not until the next chapter that we shall begin to consider how Java may be used in network programming. If the meaning of any term covered here is not clear when that term is later encountered in context, the reader should refer back to this chapter to refresh his/her memory.

It would be very easy to make this chapter considerably larger than it currently is, simply by including a great deal of dry, technical material that would be unlikely to be of any practical use to the intended readers of this book. However, this chapter is intentionally brief, the author having avoided the inclusion of material that is not of relevance to the use of Java for network programming. The reader who already has a sound grasp of network concepts may safely skip this chapter entirely.

1.1 Clients, Servers and Peers

The most common categories of network software nowadays are *clients* and *servers*. These two categories have a symbiotic relationship and the term *client/server programming* has become very widely used in recent years. It is important to distinguish firstly between a server and the machine upon which the server is running (called the *host* machine), since I.T. workers often refer loosely to the host machine as 'the server'. Though this common usage has no detrimental practical effects for the majority of I.T. tasks, those I.T. personnel who are unaware of the distinction and subsequently undertake network programming are likely to be caused a significant amount of conceptual confusion until this distinction is made known to them.

A server, as the name implies, provides a service of some kind. This service is provided for clients that connect to the server's host machine specifically for the purpose of accessing the service. Thus, it is the clients that initiate a dialogue with the server. (These clients, of course, are also programs and are **not** human clients!) Common services provided by such servers include the 'serving up' of Web pages

(by Web servers) and the downloading of files from servers' host machines via the File Transfer Protocol (FTP servers). For the former service, the corresponding client programs would be Web browsers (such as Netscape Communicator or Microsoft Explorer). Though a client and its corresponding server will normally run on different machines in a real-world application, it is perfectly possible for such programs to run on the *same* machine. Indeed, it is often very convenient (as will be seen in subsequent chapters) for server and client(s) to be run on the same machine, since this provides a very convenient 'sandbox' within which such applications may be tested before being released (or, more likely, before final testing on separate machines). This avoids the need for multiple machines and multiple testing personnel.

In some applications, such as messaging services, it is possible for programs on users' machines to communicate directly with each other in what is called *peer-to-peer* (or *P2P*) mode. However, for many applications, this is either not possible or prohibitively costly in terms of the number of simultaneous connections required. For example, the World Wide Web simply does not allow clients to communicate directly with each other. However, some applications use a server as an intermediary, in order to provide 'simulated ' peer-to-peer facilities. Alternatively, both ends of the dialogue may act as both client and server. Peer-to-peer systems are beyond the intended scope of this text, though, and no further mention will be made of them.

1.2 Ports and Sockets

These entities lie at the heart of network communications. For anybody not already familiar with the use of these terms in a network programming context, the two words very probably conjure up images of hardware components. However, although they are closely associated with the hardware communication links between computers within a network, *ports* and *sockets* are not themselves hardware elements, but abstract concepts that allow the programmer to make use of those communication links.

A port is a *logical* connection to a computer (as opposed to a physical connection) and is identified by a number in the range 1-65535. This number has no correspondence with the number of physical connections to the computer, of which there may be only one (even though the number of ports used on that machine may be much greater than this). Ports are implemented upon all computers attached to a network, but it is only those machines that have server programs running on them for which the network programmer will refer explicitly to port numbers. Each port may be dedicated to a particular server/service (though the number of available ports will normally greatly exceed the number that is actually used). Port numbers in the range 1-1023 are normally set aside for the use of specified standard services, often referred to as 'well-known' services. For example, port 80 is normally used by Web servers. Some of the more common well-known services are listed in Section 1.4. Application programs wishing to use ports for non-standard services should avoid using port numbers 1-1023. (A range of 1024-65535 should be more than enough for even the most prolific of network programmers!)

For each port supplying a service, there is a server program waiting for any requests. All such programs run together in parallel on the host machine. When a client attempts to make connection with a particular server program, it supplies the port number of the associated service. The host machine examines the port number and passes the client's transmission to the appropriate server program for processing.

In most applications, of course, there are likely to be multiple clients wanting the same service at the same time. A common example of this requirement is that of multiple browsers (quite possibly thousands of them) wanting Web pages from the same server. The server, of course, needs some way of distinguishing between clients and keeping their dialogues separate from each other. This is achieved via the use of *sockets*. As stated earlier, a socket is an abstract concept and **not** an element of computer hardware. It is used to indicate one of the two end-points of a communication link between two processes. When a client wishes to make connection to a server, it will create a socket at its end of the communication link. Upon receiving the client's initial request (on a particular port number), the server will create a new socket at its end that will be dedicated to communication with that particular client. Just as one hardware link to a server may be associated with many ports, so too may one port be associated with many sockets. More will be said about sockets in Chapter 2.

1.3 The Internet and IP Addresses

An internet (lower-case 'i') is a collection of computer networks that allows any computer on any of the associated networks to communicate with any other computer located on any of the other associated networks (or on the same network, of course). The protocol used for such communication is called the Internet Protocol (IP). *The* Internet (upper-case 'I') is the world's largest IP-based network. Each computer on the Internet has a unique IP address, the current version of which is IPv4 (Internet Protocol version 4). This represents machine addresses in what is called **quad notation**. This is made up of four eight-bit numbers (i.e., numbers in the decimal range 0-255), separated by dots. For example, 131.122.3.219 would be one such address. Due to a growing shortage of IPv4 addresses, IPv4 is due to be replaced with IPv6, the draft standard for which was published on the 10th of August, 1998. IPv6 uses 128-bit addresses, which provide massively more addresses. Many common Internet applications already work with IPv6 and it is expected that IPv6 will gradually replace IPv4, with the two coexisting for a number of years during a transition period.

Recent years have witnessed an explosion in the growth and use of the Internet. As a result, there has arisen a need for a programming language with features designed specifically for network programming. Java provides these features and does so in a platform-independent manner, which is vital for a heterogeneous network such as the Internet. Java is sometimes referred to as 'the language of the Internet' and it is the use of Java in this context that has had a major influence on the popularisation of the language. For many programmers, the need to program for the Internet is one of the main reasons, if not *the* reason, for learning to program in Java.

1.4 Internet Services, URLs and DNS

Whatever the service provided by a server, there must be some established *protocol* governing the communication that takes place between server and client. Each end of the dialogue must know what may/must be sent to the other, the format in which it should be sent, the sequence in which it must be sent (if sequence matters) and, for 'open-ended' dialogues, how the dialogue is to be terminated. For the standard services, such protocols are made available in public documents, usually by either the Internet Engineering Task Force (IETF) or the World Wide Web Consortium (W3C). Some of the more common services and their associated ports are shown in Figure 1.1. For a more esoteric or 'bespoke' service, the application writer must establish a protocol and convey it to the intended users of that service.

Protocol name	**Port number**	**Nature of service**
Echo	7	The server simply echoes the data sent to it. This is useful for testing purposes.
Daytime	13	Provides the ASCII representation of the current date and time on the server.
FTP-data	20	Transferring files. (FTP uses two ports.)
FTP	21	Sending FTP commands like PUT and GET.
Telnet	23	Remote login and command line interaction.
SMTP	25	E-mail. (Simple Mail Transfer Protocol.)
HTTP	80	HyperText Transfer Protocol (the World Wide Web protocol).
NNTP	119	Usenet. (Network News Transfer Protocol.)

Table 1.1 Some well-known network services.

A URL (Uniform Resource Locator) is a unique identifier for any resource located on the Internet. It has the following structure (in which BNF notation is used):

```
<protocol>://<hostname>[:<port>][/<pathname>][/<filename>[#<section>]]
```

For example:

```
http://java.sun.com/j2se/1.5.0/download.jsp
```

For a well-known protocol, the port number may be omitted and the default port number will be assumed. Thus, since the example above specifies the HTTP protocol (the protocol of the Web) and does not specify on which port of the host machine the service is available, it will be assumed that the service is running on port 80 (the default port for Web servers). If the file name is omitted, then the server sends a default file from the directory specified in the path name. (This default file will commonly be called *index.html* or *default.html*.) The 'section' part of the URL (not often specified) indicates a named 'anchor' in an HTML document. For example, the HTML anchor in the tag

```
<A NAME="thisPlace"></A>
```

would be referred to as `thisPlace` by the section component of the URL.

Since human beings are generally much better at remembering meaningful strings of characters than they are at remembering long strings of numbers, the Domain Name System was developed. A *domain name*, also known as a *host name*, is the user-friendly equivalent of an IP address. In the previous example of a URL, the domain name was `java.sun.com`. The individual parts of a domain name don't correspond to the individual parts of an IP address. In fact, domain names don't always have four parts (as IPv4 addresses must have).

Normally, human beings will use domain names in preference to IP addresses, but they can just as well use the corresponding IP addresses (if they know what they are!). The *Domain Name System* provides a mapping between IP addresses and domain names and is held in a distributed database. The IP address system and the DNS are governed by ICANN (the Internet Corporation for Assigned Names and Numbers), which is a non-profitmaking organisation. When a URL is submitted to a browser, the DNS automatically converts the domain name part into its numeric IP equivalent.

1.5 TCP

In common with all modern computer networks, the Internet is a **packet-switched** network, which means that messages between computers on the Internet are broken up into blocks of information called **packets**, with each packet being handled separately and possibly travelling by a completely different route from that of other such packets from the same message. IP is concerned with the **routing** of these packets through an internet. Introduced by the American military during the Cold War, it was designed from the outset to be robust. In the event of a military strike against one of the network routers, the rest of the network **had** to continue to function as normal, with messages that would have gone through the damaged router being re-routed. IP is responsible for this re-routing. It attaches the IP address of the intended recipient to each packet and then tries to determine the most efficient route available to get to the ultimate destination (taking damaged routers into account).

However, since packets could still arrive out of sequence, be corrupted or even not arrive at all (without indication to either sender or intended recipient that

anything had gone wrong), it was decided to place another protocol layer on top of IP. This further layer was provided by TCP (Transmission Control Protocol), which allowed each end of a connection to acknowledge receipt of IP packets and/or request retransmission of lost or corrupted packets. In addition, TCP allows the packets to be rearranged into their correct sequence at the receiving end. IP and TCP are the two commonest protocols used on the Internet and are almost invariably coupled together as TCP/IP. TCP is the higher level protocol that uses the lower level IP.

For Internet applications, a four-layer model is often used, which is represented diagrammatically in Figure 1.1 below. The transport layer will often comprise the TCP protocol, but may be UDP (described in the next section), while the internet layer will always be IP. Each layer of the model represents a different level of abstraction, with higher levels representing higher abstraction. Thus, although applications may appear to be communicating directly with each other, they are actually communicating directly only with their transport layers. The transport and internet layers, in their turn, communicate directly only with the layers immediately above and below them, while the host-to-network layer communicates directly only with the IP layer at each end of the connection. When a message is sent by the application layer at one end of the connection, it passes through each of the lower layers. As it does so, each layer adds further protocol data specific to the particular protocol at that level. For the TCP layer, this process involves breaking up the data packets into TCP segments and adding sequence numbers and checksums; for the IP layer, it involves placing the TCP segments into IP packets called **datagrams** and adding the routing details. The host-to-network layer then converts the digital data into an analogue form suitable for transmission over the carrier wire, sends the data and converts it back into digital form at the receiving end.

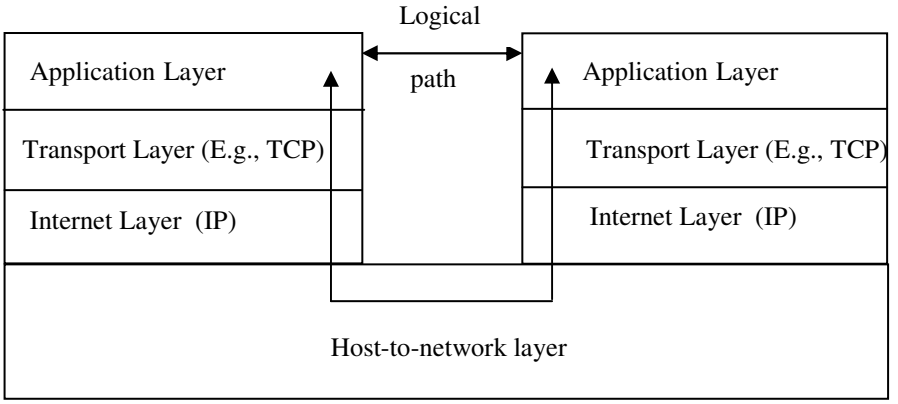

Figure 1.1 The 4-Layer Network Model

At the receiving end, the message travels up through the layers until it reaches the receiving application layer. As it does so, each layer converts the message into a form suitable for receipt by the next layer (effectively reversing the corresponding process carried out at the sending end) and carries out checks appropriate to its own

protocol. If recalculation of checksums reveals that some of the data has been corrupted or checking of sequence numbers shows that some data has not been received, then the transport layer requests re-transmission of the corrupt/missing data. Otherwise, the transport layer acknowledges receipt of the packets. All of this is completely transparent to the application layer. Once all the data has been received, converted and correctly sequenced, it is presented to the recipient application layer as though that layer had been in direct communication with the sending application layer. The latter may then send a response in exactly the same manner (and so on). In fact, since TCP provides full duplex transmission, the two ends of the connection may be sending data simultaneously.

The above description has deliberately hidden many of the low-level details of implementation, particularly the tasks carried out by the host-to-network layer. In addition, of course, the initial transmission may have passed through several routers and their associated layers before arriving at its ultimate destination. However, this high-level view covers the basic stages that are involved and is quite sufficient for our purposes.

Another network model that is often referred to is the seven-layer Open Systems Interconnection (OSI) model. However, this model is an unnecessarily complex one for our purposes and is better suited to non-TCP/IP networks anyway.

1.6 UDP

Most Internet applications use TCP as their transport mechanism. Unfortunately, the checks built into TCP to make it such a robust protocol do not come without a cost. The overhead of providing facilities such as confirmation of receipt and re-transmission of lost or corrupted packets means that TCP is a relatively slow transport mechanism. For many applications (e.g., file transfer), this does not really matter greatly. For these applications, it is much more important that the data arrives intact and in the correct sequence, both of which are guaranteed by TCP. For some applications, however, these factors are not the most important criteria and the relatively slow throughput speed provided by TCP is simply not feasible. Such applications include the playing of audio and video while the associated files are being downloaded, via what is called *streaming*. One of the most popular streaming technologies is called *RealAudio*. *RealAudio* does not use TCP, because of its large overhead. This and other such applications use User Datagram Protocol (UDP). UDP is an unreliable protocol, since:

- it doesn't guarantee that each packet will arrive;
- it doesn't guarantee that packets will be in the right order.

UDP doesn't re-send a packet if it is missing or there is some other error, and it doesn't assemble packets into the correct order. However, it is significantly *faster* than TCP. For applications such as the streaming of audio or video, losing a few bits of data is much better than waiting for re-transmission of the missing data. The major objective in these two applications is to keep playing the sound/video without

interruption. In addition, it is possible to build error-checking code into the UDP data streams to compensate for the missing data.