CHAPTER 17

NETWORK PROTOCOLS

To destroy communication completely, there must be no rules in common between transmitter and receiver—neither of alphabet nor of syntax.

—*On Human Communication,* Colin Cherry

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Explain the motivation for organizing communication functions into a layered protocol architecture.
- Describe the TCP/IP protocol architecture.
- Understand the purpose of the Sockets facility and how to use it.
- Describe the networking features in Linux.
- Understand how TFTP works.

With the increasing availability of inexpensive yet powerful personal computers and servers, there has been an increasing trend toward distributed data processing (DDP), in which processors, data, and other aspects of a data processing system may be dispersed within an organization. A DDP system involves a partitioning of the computing function and may also involve a distributed organization of databases, device control, and interaction (network) control.

In many organizations, there is heavy reliance on personal computers coupled with servers. Personal computers are used to support a variety of user-friendly applications, such as word processing, spreadsheet, and presentation graphics. The servers house the corporate database plus sophisticated database management and information systems software. Linkages are needed among the personal computers and between each personal computer and the server. Various approaches are in common use, ranging from treating the personal computer as a simple terminal to

implementing a high degree of integration between personal computer applications and the server database.

These application trends have been supported by the evolution of distributed capabilities in the operating system and supporting utilities. A spectrum of distributed capabilities has been explored:

- **Communications architecture:** This is software that supports a group of networked computers. It provides support for distributed applications, such as electronic mail, file transfer, and remote terminal access. However, the computers retain a distinct identity to the user and to the applications, which must communicate with other computers by explicit reference. Each computer has its own separate operating system, and a heterogeneous mix of computers and operating systems is possible, as long as all machines support the same communications architecture. The most widely used communications architecture is the TCP/IP protocol suite, examined in this chapter.
- **Network operating system:** This is a configuration in which there is a network of application machines, usually single-user workstations and one or more "server" machines. The server machines provide networkwide services or applications, such as file storage and printer management. Each computer has its own private operating system. The network operating system is simply an adjunct to the local operating system that allows application machines to interact with server machines. The user is aware that there are multiple independent computers and must deal with them explicitly. Typically, a common communications architecture is used to support these network applications.
- **Distributed operating system:** A common operating system shared by a network of computers. It looks to its users like an ordinary

centralized operating system but provides the user with transparent access to the resources of a number of machines. A distributed operating system may rely on a communications architecture for basic communications functions; more commonly, a stripped-down set of communications functions is incorporated into the operating system to provide efficiency.

The technology of the communications architecture is well developed and is supported by all vendors. Network operating systems are a more recent phenomena, but a number of commercial products exist. The leading edge of research and development for distributed systems is in the area of distributed operating systems. Although some commercial systems have been introduced, fully functional distributed operating systems are still at the experimental stage.

In this chapter and the next, we provide a survey of distributed processing capabilities. This chapter focuses on the underlying network protocol software.

17.1 THE NEED FOR A PROTOCOL ARCHITECTURE

When computers, terminals, and/or other data processing devices exchange data, the procedures involved can be quite complex. Consider, for example, the transfer of a file between two computers. There must be a data path between the two computers, either directly or via a communication network. But more is needed. Typical tasks to be performed include the following:

1. The source system must either activate the direct data communication path or inform the communication network of the identity of the desired destination system.

2. The source system must ascertain that the destination system is prepared to receive data.
3. The file transfer application on the source system must ascertain that the file management program on the destination system is prepared to accept and store the file for this particular user.
4. If the file formats or data representations used on the two systems are incompatible, one or the other system must perform a format translation function.

The exchange of information between computers for the purpose of cooperative action is generally referred to as *computer communications*. Similarly, when two or more computers are interconnected via a communication network, the set of computer stations is referred to as a *computer network*. Because a similar level of cooperation is required between a terminal and a computer, these terms are often used when some of the communicating entities are terminals.

In discussing computer communications and computer networks, two concepts are paramount:

- Protocols
- Computer communications architecture, or protocol architecture

A **protocol** is used for communication between entities in different systems. The terms *entity* and *system* are used in a very general sense. Examples of entities are user application programs, file transfer packages, database management systems, electronic mail facilities, and terminals. Examples of systems are computers, terminals, and remote sensors. Note that in some cases the entity and the system in which it resides are coextensive (e.g., terminals). In general, an entity is anything capable of

sending or receiving information, and a system is a physically distinct object that contains one or more entities. For two entities to communicate successfully, they must "speak the same language." What is communicated, how it is communicated, and when it is communicated must conform to mutually agreed conventions between the entities involved. The conventions are referred to as a protocol, which may be defined as a set of rules governing the exchange of data between two entities. The key elements of a protocol are as follows:

- **Syntax:** Includes such things as data format and signal levels
- **Semantics:** Includes control information for coordination and error handling
- **Timing:** Includes speed matching and sequencing

Appendix 17A provides a specific example of a protocol, the Internet standard Trivial File Transfer Protocol (TFTP).

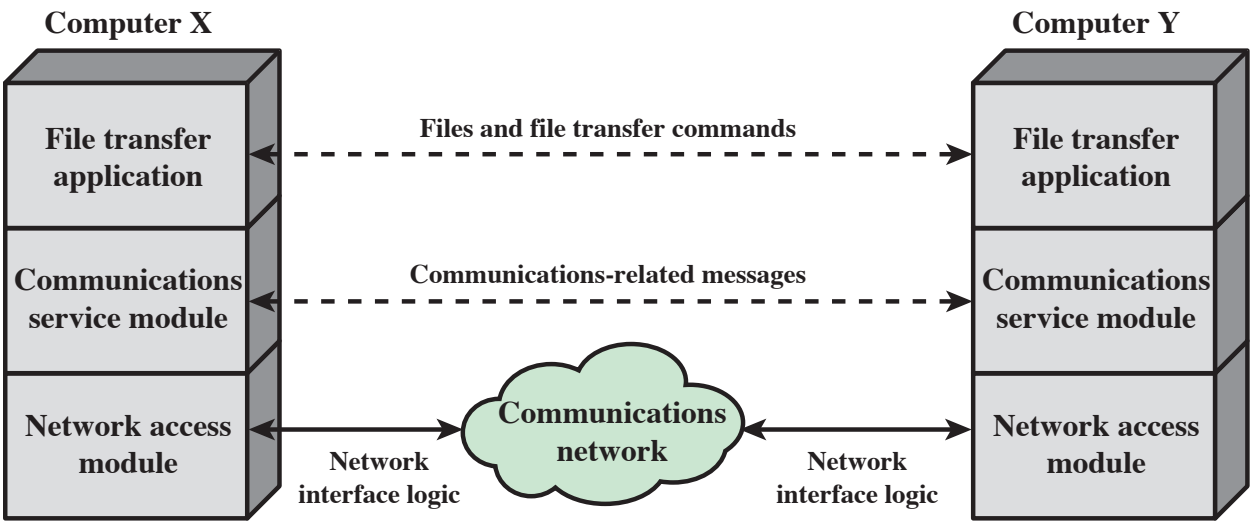

Figure 17.1 A Simplified Architecture for File Transfer

Having introduced the concept of a protocol, we can now introduce the concept of a **protocol architecture**. It is clear that there must be a high degree of cooperation between the two computer systems. Instead of implementing the logic for this as a single module, the task is broken up into subtasks, each of which is implemented separately. As an example, Figure 17.1 suggests the way in which a file transfer facility could be implemented. Three modules are used. Tasks 3 and 4 in the preceding list could be performed by a file transfer module. The two modules on the two systems exchange files and commands. However, rather than requiring the file transfer module to deal with the details of actually transferring data and commands, the file transfer modules each rely on a communications service module. This module is responsible for making sure that the file transfer commands and data are reliably exchanged between systems. The manner in which a communications service module functions is explored subsequently. Among other things, this module would perform task 2. Finally, the nature of the exchange between the two communications service

modules is independent of the nature of the network that interconnects them. Therefore, rather than building details of the network interface into the communications service module, it makes sense to have a third module, a network access module, that performs task 1 by interacting with the network.

To summarize, the file transfer module contains all the logic that is unique to the file transfer application, such as transmitting passwords, file commands, and file records. These files and commands must be transmitted reliably. However, the same sorts of reliability requirements are relevant to a variety of applications (e.g., electronic mail, document transfer). Therefore, these requirements are met by a separate communications service module that can be used by a variety of applications. The communications service module is concerned with assuring that the two computer systems are active and ready for data transfer and for keeping track of the data that are being exchanged to assure delivery. However, these tasks are independent of the type of network that is being used. Therefore, the logic for actually dealing with the network is put into a separate network access module. If the network to be used is changed, only the network access module is affected.

Thus, instead of a single module for performing communications, there is a structured set of modules that implements the communications function. That structure is referred to as a protocol architecture. An analogy might be useful at this point. Suppose an executive in office X wishes to send a document to an executive in office Y. The executive in X prepares the document and perhaps attaches a note. This corresponds to the actions of the file transfer application in Figure 17.1. Then the executive in X hands the document to a secretary or administrative assistant (AA). The AA in X puts the document in an envelope and puts Y's address and X's return address on the outside. Perhaps the envelope is also marked "confidential." The AA's

actions correspond to the communications service module in Figure 17.1. The AA in X then gives the package to the shipping department. Someone in the shipping department decides how to send the package: mail, UPS, or express courier. The shipping department attaches the appropriate postage or shipping documents to the package and ships it out. The shipping department corresponds to the network access module of Figure 17.1. When the package arrives at Y, a similar layered set of actions occurs. The shipping department at Y receives the package and delivers it to the appropriate AA or secretary based on the name on the package. The AA opens the package and hands the enclosed document to the executive to whom it is addressed.

17.2 THE TCP/IP PROTOCOL ARCHITECTURE

The TCP/IP protocol architecture is a result of protocol research and development conducted on the experimental packet-switched network, ARPANET, funded by the Defense Advanced Research Projects Agency (DARPA), and is generally referred to as the TCP/IP protocol suite. This protocol suite consists of a large collection of protocols that have been issued as Internet standards by the Internet Activities Board (IAB). Appendix L provides a discussion of Internet standards.

TCP/IP Layers

In general terms, computer communications can be said to involve three agents: applications, computers, and networks. Examples of applications include file transfer and electronic mail. The applications that we are concerned with here are distributed applications that involve the exchange of data between two computer systems. These applications, and others, execute on computers that can often support multiple simultaneous

applications. Computers are connected to networks, and the data to be exchanged are transferred by the network from one computer to another. Thus, the transfer of data from one application to another involves first getting the data to the computer in which the application resides and then getting the data to the intended application within the computer.

There is no official TCP/IP protocol model. However, based on the protocol standards that have been developed, we can organize the communication task for TCP/IP into five relatively independent layers, from bottom to top:

- Physical layer
- Network access layer
- Internet layer
- Host-to-host, or transport layer
- Application layer

The **physical layer** covers the physical interface between a data transmission device (e.g., workstation, computer) and a transmission medium or network. This layer is concerned with specifying the characteristics of the transmission medium, the nature of the signals, the data rate, and related matters.

The **network access layer** is concerned with the exchange of data between an end system (server, workstation, etc.) and the network to which it is attached. The sending computer must provide the network with the address of the destination computer, so that the network may route the data to the appropriate destination. The sending computer may wish to invoke certain services, such as priority, that might be provided by the network. The specific software used at this layer depends on the type of network to be used; different standards have been developed for circuit switching, packet

switching (e.g., frame relay), LANs (e.g., Ethernet), and others. Thus it makes sense to separate those functions having to do with network access into a separate layer. By doing this, the remainder of the communications software, above the network access layer, need not be concerned about the specifics of the network to be used. The same higher-layer software should function properly regardless of the particular network to which the computer is attached.

The network access layer is concerned with access to and routing data across a network for two end systems attached to the same network. In those cases where two devices are attached to different networks, procedures are needed to allow data to traverse multiple interconnected networks. This is the function of the internet layer. The **Internet Protocol (IP)** is used at this layer to provide the routing function across multiple networks. This protocol is implemented not only in the end systems but also in routers. A **router** is a processor that connects two networks and whose primary function is to relay data from one network to the other on a route from the source to the destination end system.

Regardless of the nature of the applications that are exchanging data, there is usually a requirement that data be exchanged reliably. That is, we would like to be assured that all the data arrive at the destination application and that the data arrive in the same order in which they were sent. As we shall see, the mechanisms for providing reliability are essentially independent of the nature of the applications. Thus, it makes sense to collect those mechanisms in a common layer shared by all applications; this is referred to as the host-to-host layer, or **transport layer**. The Transmission Control Protocol (TCP) is the most commonly used protocol to provide this functionality.

Finally, the **application layer** contains the logic needed to support the various user applications. For each different type of application, such as file transfer, a separate module is needed that is peculiar to that application.

TCP and UDP

For most applications running as part of the TCP/IP protocol architecture, the transport layer protocol is TCP. TCP provides a reliable connection for the transfer of data between applications. A connection is simply a temporary logical association between two entities in different systems. For the duration of the connection, each entity keeps track of segments coming and going to the other entity, in order to regulate the flow of segments and to recover from lost or damaged segments.

Figure 17.2a shows the header format for TCP, which is a minimum of 20 octets, or 160 bits. The Source Port and Destination Port fields identify the applications at the source and destination systems that are using this connection. The Sequence Number, Acknowledgment Number, and Window fields provide flow control and error control. The checksum is a 16-bit code based on the contents of the segment used to detect errors in the TCP segment.

In addition to TCP, there is one other transport-level protocol that is in common use as part of the TCP/IP protocol suite: the User Datagram Protocol (UDP). UDP does not guarantee delivery, preservation of sequence, or protection against duplication. UDP enables a process to send messages to other processes with a minimum of protocol mechanism. Some transaction-oriented applications make use of UDP; one example is SNMP (Simple Network Management Protocol), the standard network management protocol for TCP/IP networks. Because it is connectionless, UDP has very little to do. Essentially, it adds a port addressing capability to IP. This is best seen by examining the UDP header, shown in Figure 17.2b.

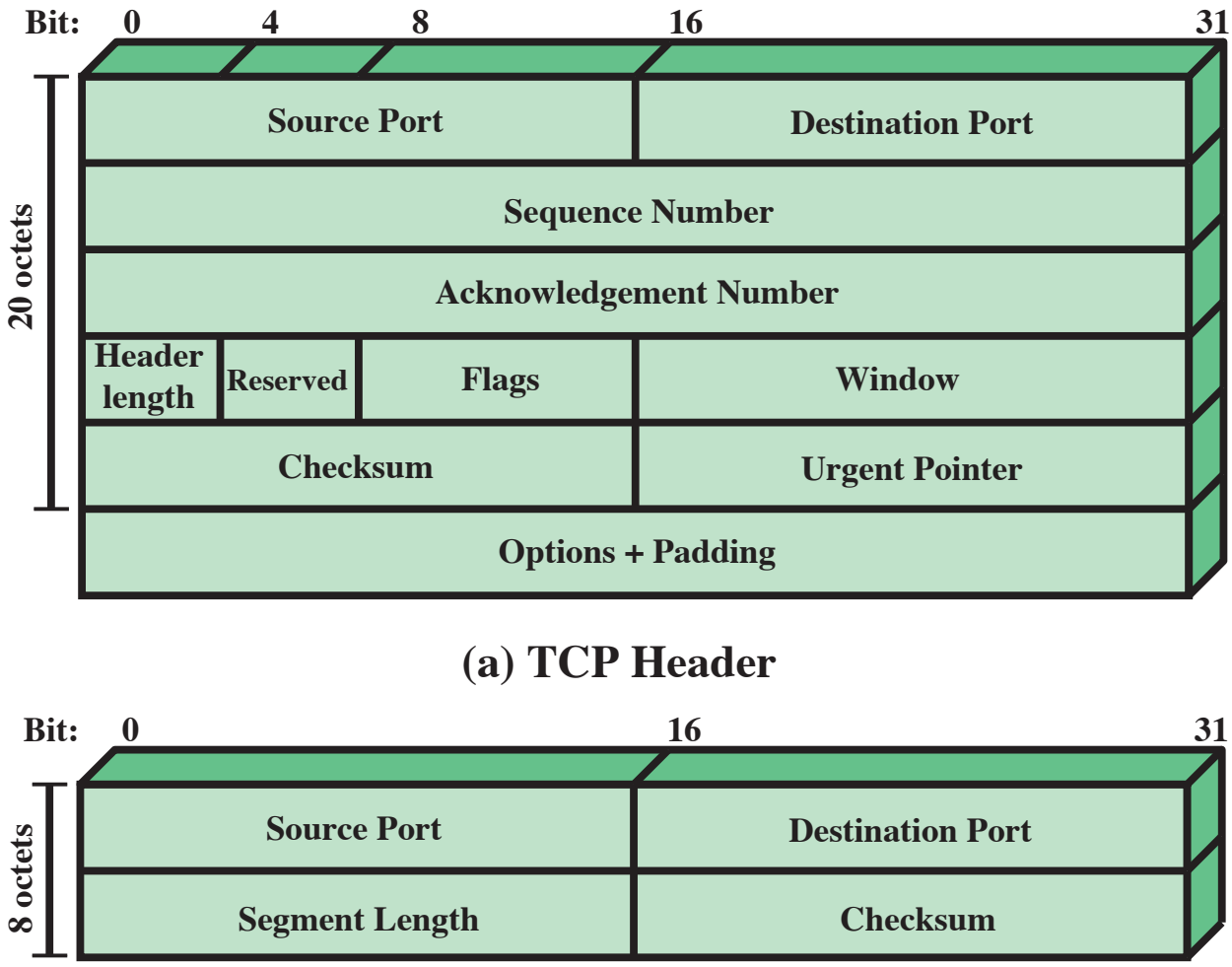

Figure 17.2 TCP and UDP Headers

IP and IPv6

For decades, the keystone of the TCP/IP protocol architecture has been IP. Figure 17.3a shows the IP header format, which is a minimum of 20 octets, or 160 bits. The header, together with the segment from the transport layer, form an IP-level block referred to as an IP datagram or an IP packet. The header includes 32-bit source and destination addresses. The Header Checksum field is used to detect errors in the header to avoid misdelivery. The Protocol field indicates whether TCP, UDP, or some other higher-layer protocol is using IP. The ID, Flags, and Fragment Offset fields are used in

the fragmentation and reassembly process, in which a single IP datagram is divided into multiple IP datagrams on transmission and then reassembled at the destination.

In 1995, the Internet Engineering Task Force (IETF), which develops protocol standards for the Internet, issued a specification for a next-generation IP, known then as IPng. This specification was turned into a standard in 1996 known as IPv6. IPv6 provides a number of functional enhancements over the existing IP, designed to accommodate the higher speeds of today's networks and the mix of data streams, including graphic and video, which are becoming more prevalent. But the driving force behind the development of the new protocol was the need for more addresses. The current IP uses a 32-bit address to specify a source or destination. With the explosive growth of the Internet and of private networks attached to the Internet, this address length became insufficient to accommodate all systems needing addresses. As Figure 17.3b shows, IPv6 includes 128-bit source and destination address fields.

Ultimately, all installations using TCP/IP are expected to migrate from the current IP to IPv6, but this process will take many years, if not decades.

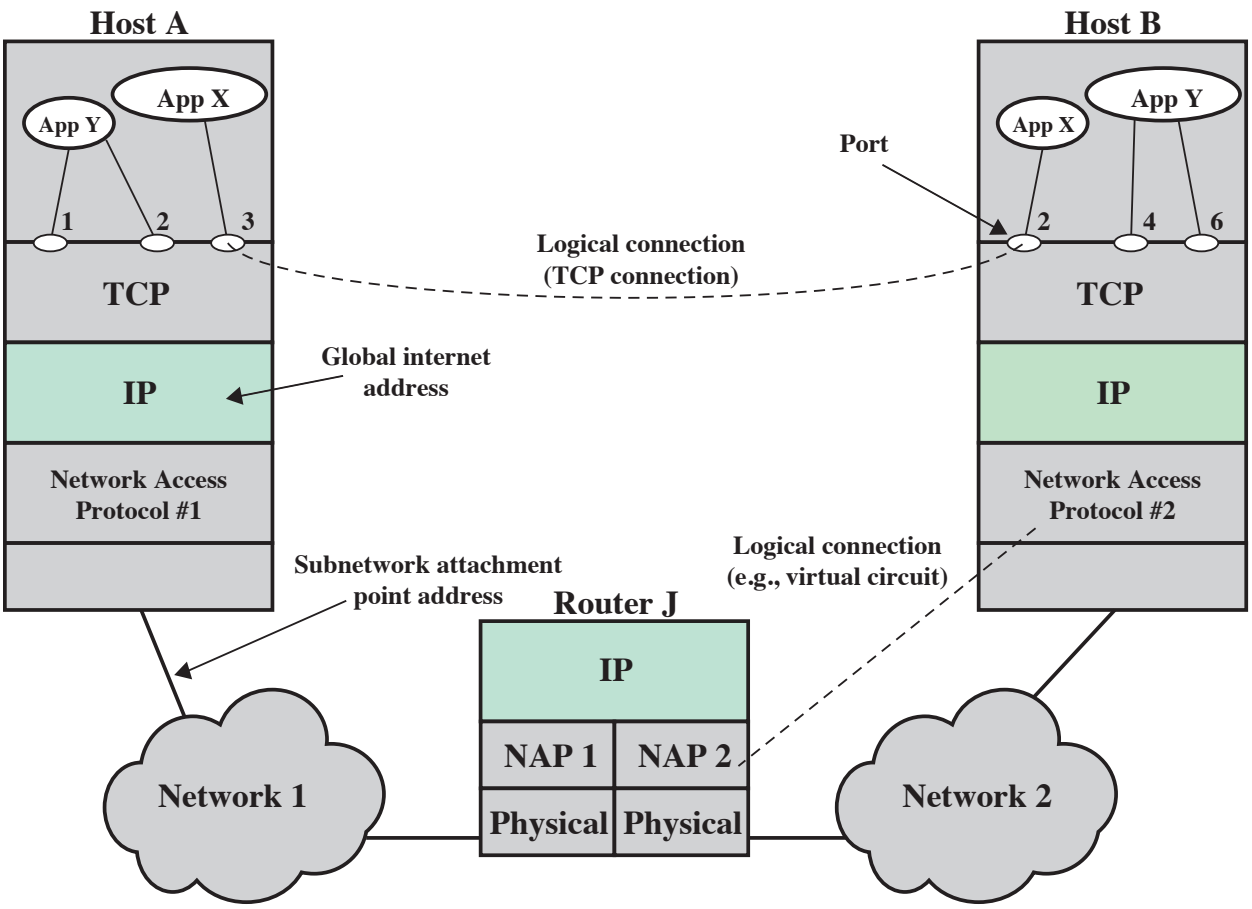

Figure 17.4 TCP/IP Concepts

Operation of TCP/IP

Figure 17.4 indicates how these protocols are configured for communications. Some sort of network access protocol, such as the Ethernet logic, is used to connect a computer to a network. This protocol enables the host to send data across the network to another host or, in the case of a host on another network, to a router. IP is implemented in all end systems and routers. It acts as a relay to move a block of data from one host, through one or more routers, to another host. TCP is implemented only in the end systems; it keeps track of the blocks of data being transferred to assure that all are delivered reliably to the appropriate application.

For successful communication, every entity in the overall system must have a unique address. In fact, two levels of addressing are needed. Each host on a network must have a unique global internet address; this allows the data to be delivered to the proper host. This address is used by IP for routing and delivery. Each application within a host must have an address that is unique within the host; this allows the host-to-host protocol (TCP) to deliver data to the proper process. These latter addresses are known as **ports**.

Let us trace a simple operation. Suppose that a process, associated with port 3 at host A, wishes to send a message to another process, associated with port 2 at host B. The process at A hands the message down to TCP with instructions to send it to host B, port 2. TCP hands the message down to IP with instructions to send it to host B. Note that IP need not be told the identity of the destination port. All it needs to know is that the data are intended for host B. Next, IP hands the message down to the network access layer (e.g., Ethernet logic) with instructions to send it to router J (the first hop on the way to B).

To control this operation, control information as well as user data must be transmitted, as suggested in Figure 17.5. Let us say that the sending process generates a block of data and passes this to TCP. TCP may break this block into smaller pieces to make it more manageable. To each of these pieces, TCP appends control information known as the TCP header (Figure 17.2a), forming a **TCP segment**. The control information is to be used by the peer TCP protocol entity at host B.

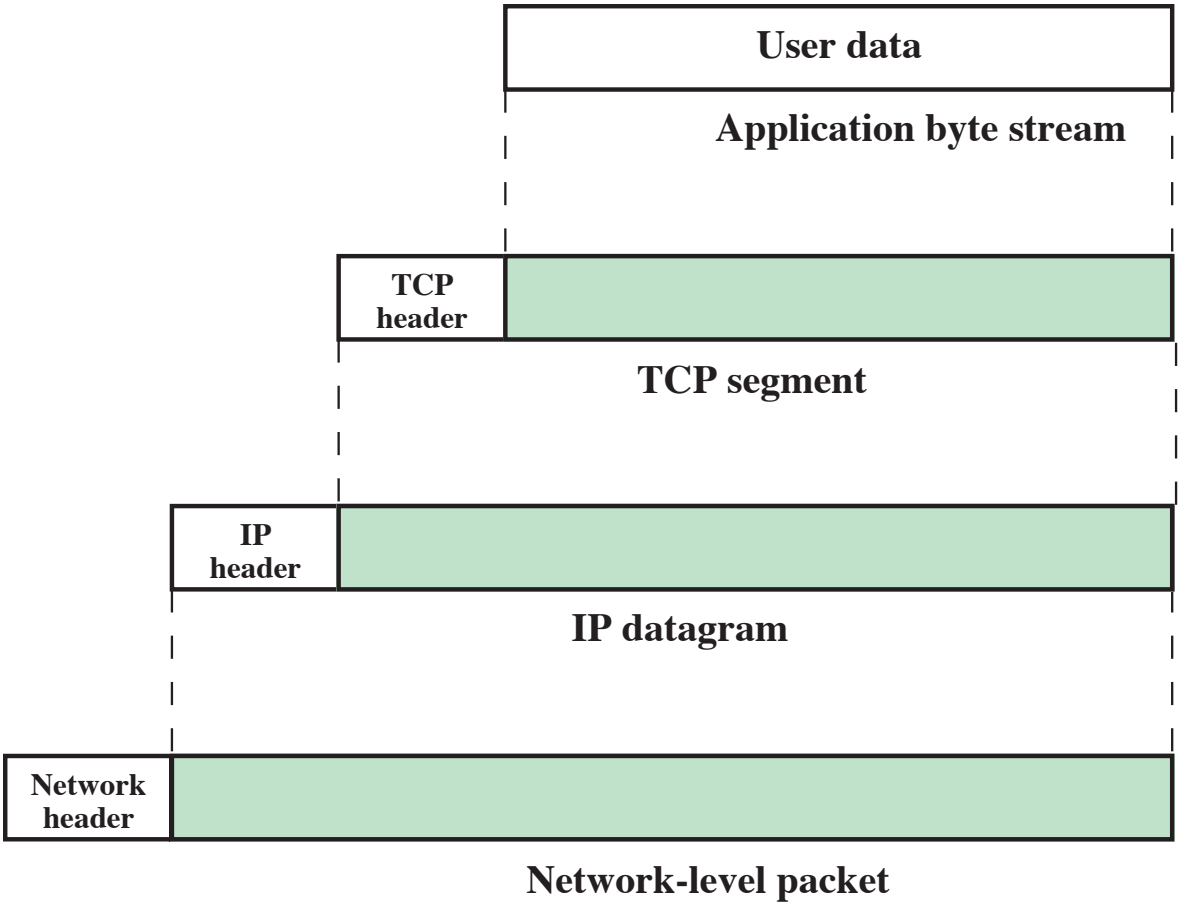

Figure 17.5 Protocol Data Units (PDUs) in the TCP/IP Architecture

Next, TCP hands each segment over to IP, with instructions to transmit it to B. These segments must be transmitted across one or more networks and relayed through one or more intermediate routers. This operation, too, requires the use of control information. Thus IP appends a header of control information (Figure 17.3) to each segment to form an **IP datagram**. An example of an item stored in the IP header is the destination host address (in this example, B).

Finally, each IP datagram is presented to the network access layer for transmission across the first network in its journey to the destination. The network access layer appends its own header, creating a packet, or frame. The packet is transmitted across the network to router J. The packet header

contains the information that the network needs in order to transfer the data across the network. Examples of items that may be contained in this header include:

- **Destination network address:** The network must know to which attached device the packet is to be delivered, in this case router J.
- **Facilities requests:** The network access protocol might request the use of certain network facilities, such as priority.

At router J, the packet header is stripped off and the IP header examined. On the basis of the destination address information in the IP header, the IP module in the router directs the datagram across network 2 to B. To do this, the datagram is again augmented with a network access header.

When the data are received at B, the reverse process occurs. At each layer, the corresponding header is removed, and the remainder is passed on to the next higher layer, until the original user data are delivered to the destination process.

TCP/IP Applications

A number of applications have been standardized to operate on top of TCP. We mention three of the most common here.

The **Simple Mail Transfer Protocol (SMTP)** provides a basic electronic mail facility. It provides a mechanism for transferring messages among separate hosts. Features of SMTP include mailing lists, return receipts, and forwarding. The SMTP protocol does not specify the way in which messages are to be created; some local editing or native electronic mail facility is required. Once a message is created, SMTP accepts the message and makes use of TCP to send it to an SMTP module on another

host. The target SMTP module will make use of a local electronic mail package to store the incoming message in a user's mailbox.

The **File Transfer Protocol (FTP)** is used to send files from one system to another under user command. Both text and binary files are accommodated, and the protocol provides features for controlling user access. When a user wishes to engage in file transfer, FTP sets up a TCP connection to the target system for the exchange of control messages. This connection allows user ID and password to be transmitted and allows the user to specify the file and file actions desired. Once a file transfer is approved, a second TCP connection is set up for the data transfer. The file is transferred over the data connection, without the overhead of any headers or control information at the application level. When the transfer is complete, the control connection is used to signal the completion and to accept new file transfer commands.

SSH (Secure Shell) provides a secure remote logon capability, which enables a user at a terminal or personal computer to log on to a remote computer and function as if directly connected to that computer. SSH also supports file transfer between the local host and a remote server. SSH enables the user and the remote server to authenticate each other; it also encrypts all traffic in both directions. SSH traffic is carried on a TCP connection.

17.3 SOCKETS[1]

The concept of sockets and sockets programming was developed in the 1980s in the UNIX environment as the Berkeley Sockets Interface. In essence, a socket enables communication between a client and server process and may be either connection oriented or connectionless. A socket

[1] This section provides a Sockets overview. Appendix M contains a more detailed treatment.

can be considered an endpoint in a communication. A client socket in one computer uses an address to call a server socket on another computer. Once the appropriate sockets are engaged, the two computers can exchange data.

Typically, computers with server sockets keep a TCP or UDP port open, ready for unscheduled incoming calls. The client typically determines the socket identification of the desired server by finding it in a Domain Name System (DNS) database. Once a connection is made, the server switches the dialogue to a different port number to free up the main port number for additional incoming calls.

Internet applications, such as TELNET and remote login (rlogin), make use of sockets, with the details hidden from the user. However, sockets can be constructed from within a program (in a language such as C or Java), enabling the programmer to easily support networking functions and applications. The sockets programming mechanism includes sufficient semantics to permit unrelated processes on different hosts to communicate.

The Berkeley Sockets Interface is the de facto standard **application programming interface (API)** for developing networking applications, spanning a wide range of operating systems. Windows Sockets (WinSock) is based on the Berkeley specification. The sockets API provides generic access to interprocess communications services. Thus, the sockets capability is ideally suited for students to learn the principles of protocols and distributed applications by hands-on program development.

The Socket

Recall that each TCP and UDP header includes source port and destination port fields (Figure 17.2). These port values identify the respective users (applications) of the two TCP entities. Also, each IPv4 and IPv6 header includes source address and destination address fields (Figure 17.3); these **IP addresses** identify the respective host systems. The concatenation of a

port value and an IP address forms a **socket**, which is unique throughout the Internet. Thus, in Figure 17.4, the combination of the IP address for host B and the port number for application X uniquely identifies the socket location of application X in host B. As the figure indicates, an application may have multiple socket addresses, one for each port into the application.

The socket is used to define an API, which is a generic communication interface for writing programs that use TCP or UDP. In practice, when used as an API, a socket is identified by the triple (protocol, local address, local process). The local address is an IP address and the local process is a port number. Because port numbers are unique within a system, the port number implies the protocol (TCP or UDP). However, for clarity and ease of implementation, sockets used for an API include the protocol as well as the IP address and port number in defining a unique socket.

Corresponding to the two protocols, the Sockets API recognizes two types of sockets: stream sockets and datagram sockets. **Stream sockets** make use of TCP, which provides a connection-oriented reliable data transfer. Therefore, with stream sockets, all blocks of data sent between a pair of sockets are guaranteed for delivery and arrive in the order that they were sent. **Datagram sockets** make use of UDP, which does not provide the connection-oriented features of TCP. Therefore, with datagram sockets, delivery is not guaranteed, nor is order necessarily preserved.

There is a third type of socket provided by the Sockets API: raw sockets. **Raw sockets** allow direct access to lower-layer protocols, such as IP.

Socket Interface Calls

This subsection summarizes the key system calls.

Socket Setup

The first step in using Sockets is to create a new socket using the `socket()` command. This command includes three parameters, the protocol family is always `PF_INET`, for the TCP/IP protocol suite. *Type* specifies whether this is a stream or datagram socket, and *protocol* specifies either TCP or UDP. The reason that both *type* and *protocol* need to be specified is to allow additional transport-level protocols to be included in a future implementation. Thus, there might be more than one datagram-style transport protocol or more than one connection-oriented transport protocol. The `socket()` command returns an integer result that identifies this socket; it is similar to a UNIX file descriptor. The exact socket data structure depends on the implementation. It includes the source port and IP address and, if a connection is open or pending, the destination port and IP address and various options and parameters associated with the connection.

After a socket is created, it must have an address to listen to. The bind() function binds a socket to a socket address. The address has the structure

```
struct sockaddr_in {
     short int sin_family;          // Address family (TCP/IP)
     unsigned short int sin_port;   // Port number
     struct in_addr sin_addr;       // Internet address
     unsigned char sin_zero[8];     // Same size as struct sockaddr
};
```

SOCKET CONNECTION

For a stream socket, once the socket is created, a connection must be set up to a remote socket. One side functions as a client and requests a connection to the other side, which acts as a server.

The server side of a connection setup requires two steps. First, a server application issues a `listen()`, indicating that the given socket is ready to accept incoming connections. The parameter *backlog* is the number of connections allowed on the incoming queue. Each incoming connections is

placed in this queue until a matching `accept()` is issued by the server side. Next, the `accept()` call is used to remove one request from the queue. If the queue is empty, the `accept()` blocks the process until a connection request arrives. If there is a waiting call, then `accept()` returns a new file descriptor for the connection. This creates a new socket, which has the IP address and port number of the remote party, the IP address of this system, and a new port number. The reason that a new socket with a new port number is assigned is that this enables the local application to continue to listen for more requests. As a result, an application may have multiple connections active at any time, each with a different local port number. This new port number is returned across the TCP connection to the requesting system.

A client application issues a `connect()` that specifies both a local socket and the address of a remote socket. If the connection attempt is unsuccessful `connect()` returns the value –1. If the attempt is successful, `connect()` returns a 0 and fills in the file descriptor parameter to include the IP address and port number of the local and foreign sockets. Recall that the remote port number may differ from that specified in the `foreignAddress` parameter because the port number is changed on the remote host.

Once a connection is set up, `getpeername()` can be used to find out who is on the other end of the connected stream socket. The function returns a value in the `sockfd` parameter.

SOCKET COMMUNICATION

For **stream communication**, the functions `send()` and `recv()` are used to send or receive data over the connection identified by the sockfd parameter. In the `send()` call, the `*msg` parameter points to the block of data to be sent and the `len` parameter specifies the number of bytes to be sent. The `flags` parameter contains control flags, typically set to 0. The

`send()` call returns the number of bytes sent, which may be less than the number specified in the len parameter. In the `recv()` call, the `*buf` parameter points to the buffer for storing incoming data, with an upper limit on the number of bytes set by the `len` parameter.

At any time, either side can close the connection with the `close()` call, which prevents further sends and receives. The `shutdown()` call allows the caller to terminate sending or receiving or both.

Figure 17.6 shows the interaction of the clients and server sides in setting up, using, and terminating a connection.

For **datagram communication**, the functions `sendto()` and `recvfrom()` are used. The `sendto()` call includes all the parameters of the `send()` call plus a specification of the destination address (IP address and port). Similarly, the `recvfrom()` call includes an address parameter, which is filled in when data are received.

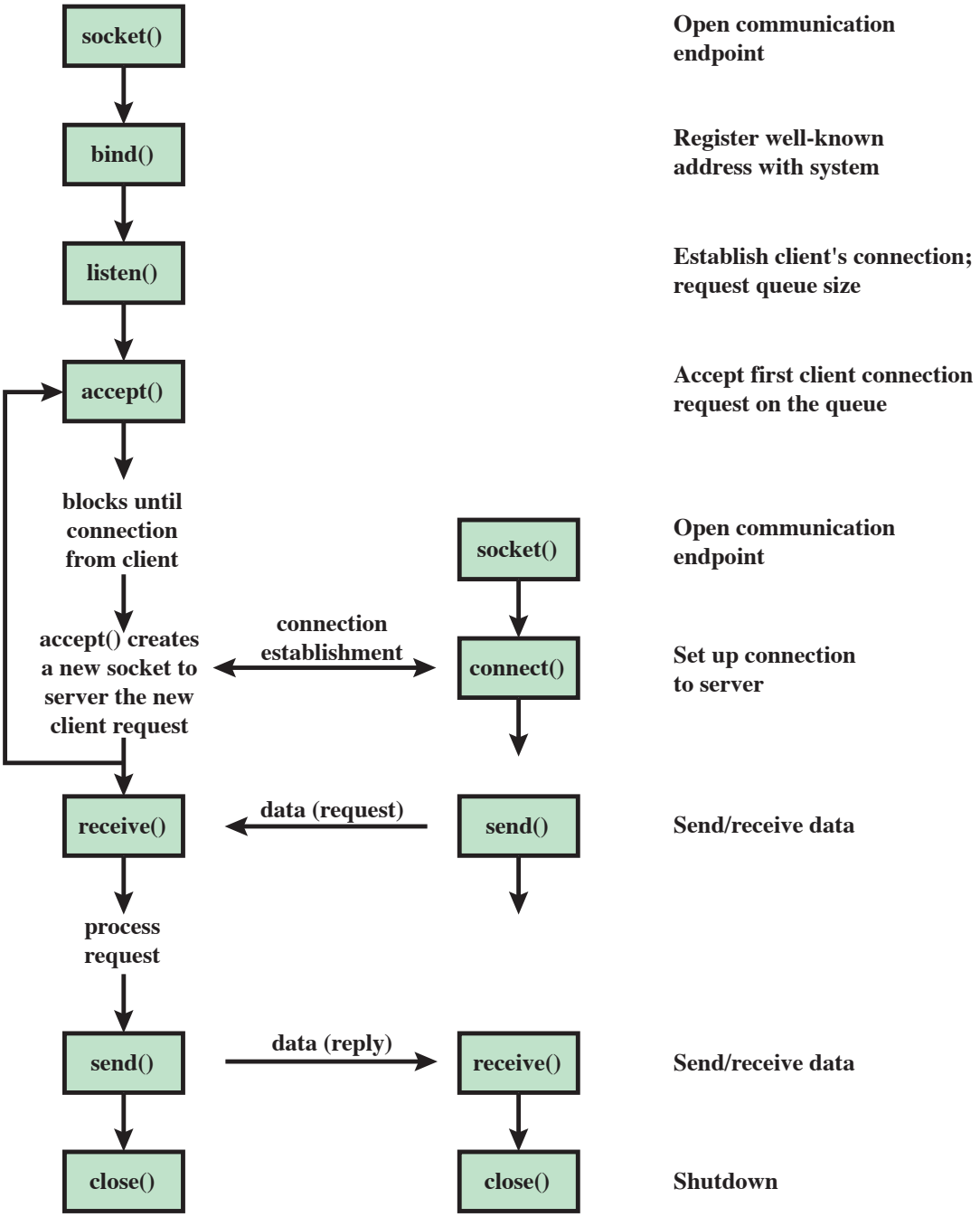

Figure 17.6 Socket System Calls for Connection-Oriented Protocol

17.4 LINUX NETWORKING

Linux supports a variety of networking architectures, in particular TCP/IP by means of Berkeley Sockets. Figure 17.7 shows the overall structure of Linux support for TCP/IP. User-level processes interact with networking devices by means of system calls to the Sockets interface. The Sockets module in turn interacts with a software package in the kernel that handles transport-layer (TCP and UDP) and IP protocol operations. This software package exchanges data with the device driver for the network interface card.

Linux implements sockets as special files. Recall from Chapter 12 that, in UNIX systems, a special file is one that contains no data but provides a mechanism to map physical devices to file names. For every new socket, the Linux kernel creates a new inode in the *sockfs* special file system.

Figure 17.7 depicts the relationships among various kernel modules involved in sending and receiving TCP/IP-based data blocks. The remainder of this section looks at the sending and receiving facilities.

Sending Data

A user process uses the sockets calls described in Section 17.3 to create new sockets, set up connections to remote sockets, and send and receive data. To send data, the user process writes data to the socket with the following file system call:

```
write(sockfd, mesg, mesglen)
```

where `mesglen` is the length of the `mesg` buffer in bytes.

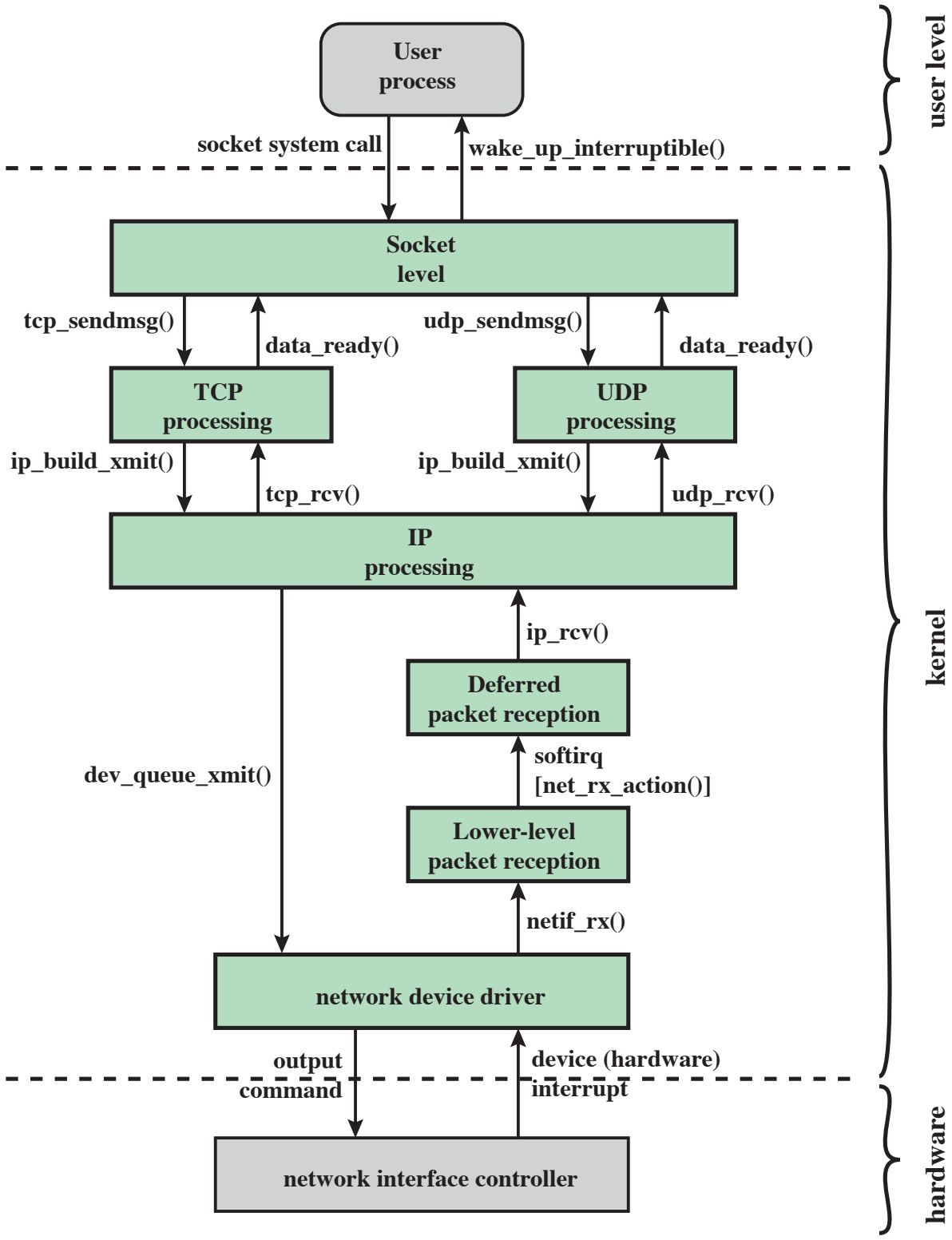

Figure 17.7 Linux Kernel Components for TCP/IP Processing

This call triggers the `write` method of the file object associated with the `sockfd` file descriptor. The file descriptor indicates whether this is a socket set up for TCP or UDP. The kernel allocates the appropriate data structures and invokes the appropriate sockets-level function to pass data to either a TCP module or a UDP module. The corresponding functions are `tcp_sendmsg()` and `udp_sendmsg()`, respectively. The transport-layer module allocates a data structure of the TCP or UPD header and performs `ip_build_xmit()` to invoke the IP-layer processing module. This module builds an IP datagram for transmission and places it in a transmission buffer for this socket. The IP-layer module then performs `dev_queue_xmit()` to queue the socket buffer for later transmission via the network device driver. When it is available, the network device driver will transmit buffered packets.

Receiving Data

Data reception is an unpredictable event and so involves the use of interrupts and deferrable functions. When an IP datagram arrives, the network interface controller issues a hardware interrupt to the corresponding network device driver. The interrupt triggers an interrupt service routine that handles the interrupt as part of the network device driver module. The driver allocates a kernel buffer for the incoming data block and transfers the data from the device controller to the buffer. The driver then performs `netif_rx()` to invoke a lower-level packet reception routine. In essence, the `netif_rx()` function places the incoming data block in a queue and then issues a soft interrupt request (`softirq`) so that the queued data will eventually be processed. The action to be performed when the `softirq` is processed is the `net_rx_action()` function.

Once a `softirq` has been queued, processing of this packet is halted until the kernel executes the `softirq` function, which is equivalent to saying

until the kernel responds to this soft interrupt request and executes the function (in this case, `net_rx_action()`) associated with this soft interrupt. There are three places in the kernel, where the kernel checks to see if any `softirqs` are pending: when a hardware interrupt has been processed, when an application-level process invokes a system call, and when a new process is scheduled for execution.

When the `net_rx_action()` function is performed, it retrieves the queued packet and passes it on to the IP packet handler by means of an `ip_rcv` call. The IP packet handler processes the IP header and then uses `tcp_rcv` or `udp_rcv` to invoke the transport-layer processing module. The transport-layer module processes the transport-layer header and passes the data to the user through the sockets interface by means of a `wake_up_interruptible()` call, which awakens the receiving process.

17.5 SUMMARY

The communication functionality required for distributed applications is quite complex. This functionality is generally implemented as a structured set of modules. The modules are arranged in a vertical, layered fashion, with each layer providing a particular portion of the needed functionality and relying on the next lower layer for more primitive functions. Such a structure is referred to as a protocol architecture.

One motivation for the use of this type of structure is that it eases the task of design and implementation. It is standard practice for any large software package to break up the functions into modules that can be designed and implemented separately. After each module is designed and implemented, it can be tested. Then the modules can be combined and tested together. This motivation has led computer vendors to develop

proprietary layered-protocol architectures. An example of this is the Systems Network Architecture (SNA) of IBM.

A layered architecture can also be used to construct a standardized set of communication protocols. In this case, the advantages of modular design remain. But, in addition, a layered architecture is particularly well suited to the development of standards. Standards can be developed simultaneously for protocols at each layer of the architecture. This breaks down the work to make it more manageable and speeds up the standards-development process. The TCP/IP protocol architecture is the standard architecture used for this purpose. This architecture contains five layers. Each layer provides a portion of the total communications function required for distributed applications. Standards have been developed for each layer. Development work continues, particularly at the top (application) layer, where new distributed applications are still being defined.

17.6 RECOMMENDED READING

[STAL14] provides a detailed description of the TCP/IP model and of the standards at each layer of the model. A very useful reference work on TCP/IP is [PARZ06], which covers the spectrum of TCP/IP-related protocols in a technically concise but thorough fashion.

An excellent concise introduction to using Sockets is [DONA01]; another good overview is [HALL01]. [MCKU05] and [WRIG95] provide details of Sockets implementation.

[BOVE06] provides good coverage of Linux networking. Another useful source is [INSO02a] and [INSO02b].

BOVE06 Bovet, D., and Cesati, M. *Understanding the Linux Kernel.* Sebastopol, CA: O'Reilly, 2006.

DONA01 Donahoo, M., and Clavert, K. *The Pocket Guide to TCP/IP Sockets.* San Francisco, CA: Morgan Kaufmann, 2001.

HALL01 Hall, B. *Beej's Guide to Network Programming Using Internet Sockets.* 2001. http://beej.us/guide/bgnet

INSO02a Insolvibile, G. "Inside the Linux Packet Filter." *Linux Journal*, February, 2002.

INSO02b Insolvibile, G. "Inside the Linux Packet Filter, Part II." *Linux Journal*, March, 2002.

MCKU05 McKusick, M., and Neville-Neil, J. *The Design and Implementation of the FreeBSD Operating System.* Reading, MA: Addison-Wesley, 2005.

PARZ06 Parziale, L., et al. *TCP/IP Tutorial and Technical Overview.* IBM Redbook GG24-3376-07, 2006. http://www.redbooks.ibm.com/abstracts/gg243376.html

STAL14 Stallings, W. *Data and Computer Communications.* Upper Saddle River: NJ: Pearson, 2014.

WRIG95 Wright, G., and Stevens, W. *TCP/IP Illustrated, Volume 2: The Implementation.* Reading, MA: Addison-Wesley, 1995.

17.7 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

application programming interface (API)	protocol	stream sockets
datagram sockets	protocol architecture	TELNET
File Transfer Protocol (FTP)	raw sockets	Transmission Control Protocol (TCP)
Internet Protocol (IP)	Simple Mail Transfer Protocol (SMTP)	User Datagram Protocol (UDP)
port	sockets	

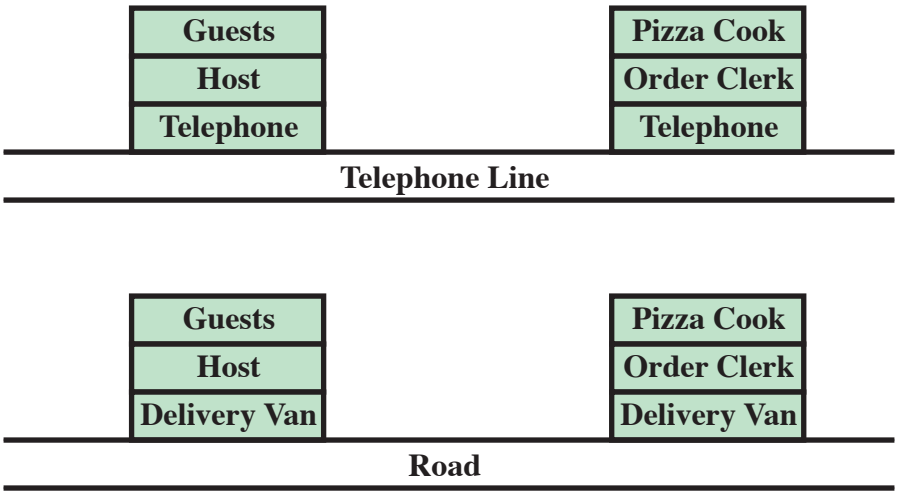

Figure 17.8 Architecture for Problem 17.1

Review Questions

17.1 What is the major function of the network access layer?

17.2 What tasks are performed by the transport layer?

17.3 What is a protocol?

17.4 What is a protocol architecture?

17.5 What is TCP/IP?

17.6 What is the purpose of the Sockets interface?

Problems

17.1 **a.** The French and Chinese prime ministers need to come to an agreement by telephone, but neither speaks the other's language. Further, neither has on hand a translator that can translate to the language of the other. However, both prime ministers have English translators on their staffs. Draw a diagram similar to Figure 17.8 to depict the situation, and describe the interaction at each layer.

b. Now suppose that the Chinese prime minister's translator can translate only into Japanese and that the French prime minister has a German translator available. A translator between German and

Japanese is available in Germany. Draw a new diagram that reflects this arrangement and describe the hypothetical phone conversation.

17.2 List the major disadvantages of the layered approach to protocols.

17.3 A TCP segment consisting of 1,500 bits of data and 160 bits of header is sent to the IP layer, which appends another 160 bits of header. This is then transmitted through two networks, each of which uses a 24-bit packet header. The destination network has a maximum packet size of 800 bits. How many bits, including headers, are delivered to the network layer protocol at the destination?

17.4 Why does the TCP header have a header length field while the UDP header does not?

17.5 The previous version of the TFTP specification, RFC 783, included the following statement:

All packets other than those used for termination are acknowledged individually unless a timeout occurs.

The new specification revises this to say

All packets other than duplicate ACKs and those used for termination are acknowledged unless a timeout occurs.

The change was made to fix a problem referred to as the "Sorcerer's Apprentice." Deduce and explain the problem.

17.6 What is the limiting factor in the time required to transfer a file using TFTP?

17.7 A user on a UNIX host wants to transfer a 4,000-byte text file to a Microsoft Windows host. In order to do this, he transfers the file by means of TFTP, using the netascii transfer mode. Even though the transfer was reported as being performed successfully, the Windows host reports the resulting file size is 4,050 bytes, rather than the original 4,000 bytes. Does this difference in the file sizes imply an error in the data transfer? Why or why not?

17.8 The TFTP specification (RFC 1350) states that the transfer identifiers (TIDs) chosen for a connection should be randomly chosen, so that the probability that the same number is chosen twice in immediate

succession is very low. What would be the problem of using the same TIDs twice in immediate succession?

17.9 In to retransmit lost packets, TFTP must keep a copy of the data it sends. How many packets of data must TFTP keep at a time to implement this retransmission mechanism?

17.10 TFTP, like most protocols, will never send an error packet in response to an error packet it receives. Why?

17.11 We have seen that in order to deal with lost packets, TFTP implements a time-out-and-retransmit scheme, by setting a retransmission timer when it transmits a packet to the remote host. Most TFTP implementations set this timer to a fixed value of about five seconds. Discuss the advantages and the disadvantages of using a fixed value for the retransmission timer.

17.12 TFTP's time-out-and-retransmission scheme implies that all data packets will eventually be received by the destination host. Will these data also be received uncorrupted? Why or why not?

17.13 This chapter mentions the use of Frame Relay as a specific protocol or system used to connect to a wide area network. Each organization will have a certain collection of services available (like Frame Relay) but this is dependent upon provider provisioning, cost and customer premises equipment. What are some of the services available to you in your area?

17.14 Wireshark is a free packet sniffer that allows you to capture traffic on a local area network. It can be used on a variety of operating systems and is available at www.ethereal.com. You must also install the WinPcap packet capture driver, which can be obtained from www.wireshark.org/.

After starting a capture from Wireshark , start a TCP-based application like TELNET, FTP, or HTTP (Web browser). Can you determine the following from your capture?

a. Source and destination layer 2 addresses (MAC)
b. Source and destination layer 3 addresses (IP)
c. Source and destination layer 4 addresses (port numbers)

17.15 Packet capture software or sniffers can be powerful management and security tools. By using the filtering capability that is built in, you can trace traffic based on several different criteria and eliminate

everything else. Use the filtering capability built into Ethereal to do the following;

a. Capture only traffic coming from your computer's MAC address.
b. Capture only traffic coming from your computer's IP address.
c. Capture only UDP-based transmissions.

APPENDIX 17A THE TRIVIAL FILE TRANSFER PROTOCOL

This appendix provides an overview of the Internet standard Trivial File Transfer Protocol (TFTP), defined in RFC 1350. Our purpose is to give the reader some flavor for the elements of a protocol. TFTP is simple enough to provide a concise example but includes most of the significant elements found in other, more complex, protocols.

Introduction to TFTP

TFTP is far simpler than the Internet standard File Transfer Protocol (FTP). There are no provisions for access control or user identification, so TFTP is only suitable for public access file directories. Because of its simplicity, TFTP is easily and compactly implemented. For example, some diskless devices use TFTP to download their firmware at boot time.

TFTP runs on top of UDP. The TFTP entity that initiates the transfer does so by sending a read or write request in a UDP segment with a destination port of 69 to the target system. This port is recognized by the target UDP module as the identifier of the TFTP module. For the duration of the transfer, each side uses a transfer identifier (TID) as its port number.

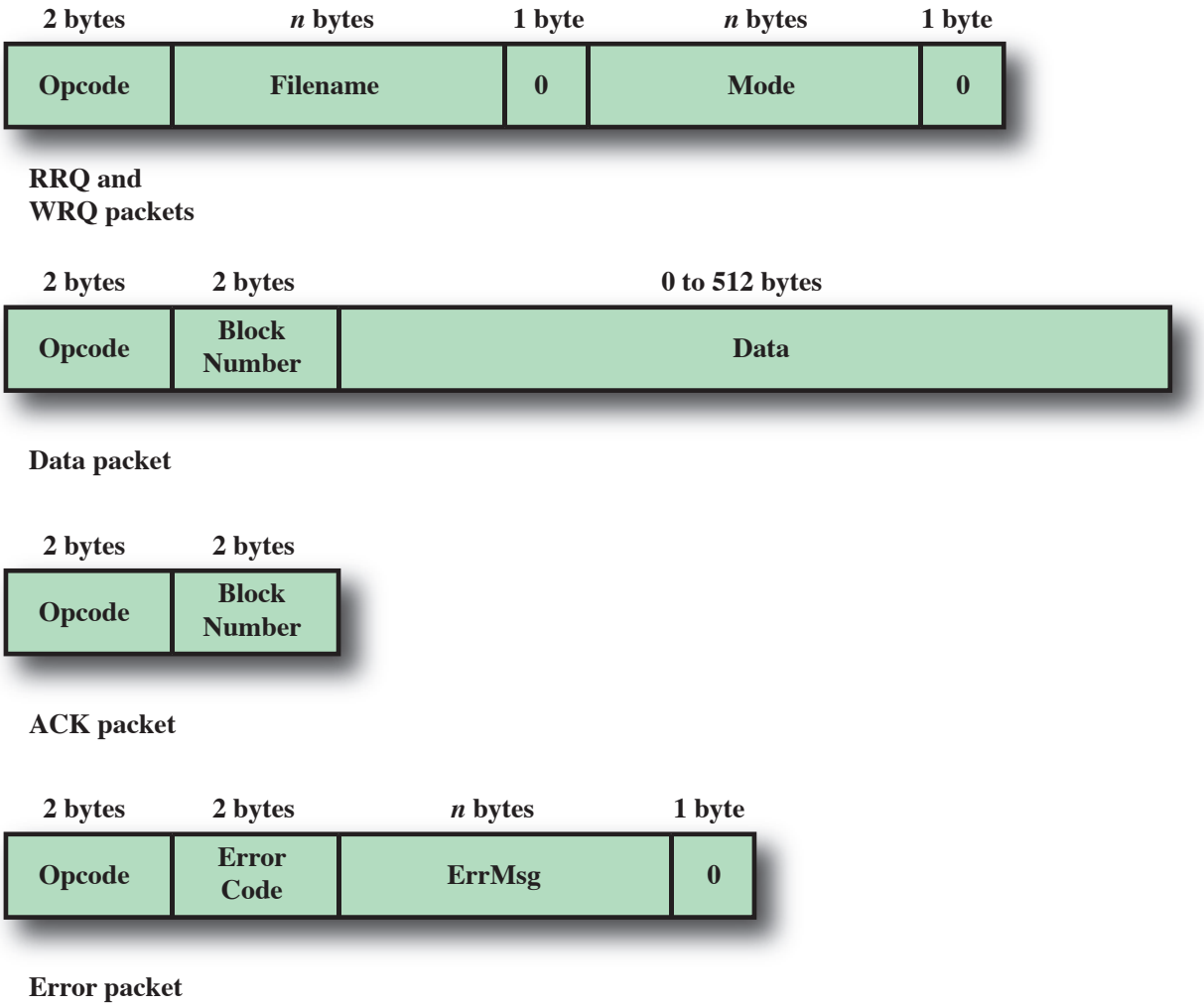

Figure 17.9 TFTP Packet Formats

TFTP Packets

TFTP entities exchange commands, responses, and file data in the form of packets, each of which is carried in the body of a UDP segment. TFTP supports five types of packets (Figure 17.9); the first two bytes contain an opcode that identifies the packet type:

- **RRQ:** The read request packet requests permission to transfer a file from the other system. The packet includes a file name, which is a

sequence of ASCII[2] bytes terminated by a zero byte. The zero byte is the means by which the receiving TFTP entity knows when the file name is terminated. The packet also includes a mode field, which indicates whether the data file is to be interpreted as a string of ASCII bytes (netascii mode) or as raw eight-bit bytes (octet mode) of data. In netascii mode, the file is transferred as lines of characters, each terminated by a carriage return, line feed. Each system must translate between its own format for character files and the TFTP format.

- **WRQ:** The write request packet requests permission to transfer a file to the other system.
- **Data:** The block numbers on data packets begin with one and increase by one for each new block of data. This convention enables the program to use a single number to discriminate between new packets and duplicates. The data field is from zero to 512 bytes long. If it is 512 bytes long, the block is not the last block of data; if it is from zero to 511 bytes long, it signals the end of the transfer.
- **ACK:** This packet is used to acknowledge receipt of a data packet or a WRQ packet. An ACK of a data packet contains the block number of the data packet being acknowledged. An ACK of a WRQ contains a block number of zero.
- **Error:** An error packet can be the acknowledgment of any other type of packet. The error code is an integer indicating the nature of the error (Table 17.1). The error message is intended for human consumption and should be in ASCII. Like all other strings, it is terminated with a zero byte.

---

[2] ASCII is the American Standard Code for Information Interchange, a standard of the American National Standards Institute. It designates a unique seven-bit pattern for each letter, with an eighth bit used for parity. ASCII is equivalent to the International Reference Alphabet (IRA), defined in ITU-T Recommendation T.50. See Appendix N for a discussion.

Table 17.1 TFTP Error Codes

Value	Meaning
0	Not defined, see error message (if any)
1	File not found
2	Access violation
3	Disk full or allocation exceeded
4	Illegal TFTP operation
5	Unknown transfer ID
6	File already exists
7	No such user

All packets other than duplicate ACKs (explained subsequently) and those used for termination are to be acknowledged. Any packet can be acknowledged by an error packet. If there are no errors, then the following conventions apply. A WRQ or a data packet is acknowledged by an ACK packet. When an RRQ is sent, the other side responds (in the absence of error) by beginning to transfer the file; thus, the first data block serves as an acknowledgment of the RRQ packet. Unless a file transfer is complete, each ACK packet from one side is followed by a data packet from the other, so that the data packet functions as an acknowledgment. An error packet can be acknowledged by any other kind of packet, depending on the circumstance.

Figure 17.10 shows a TFTP data packet in context. When such a packet is handed down to UDP, UDP adds a header to form a UDP segment. This is then passed to IP, which adds an IP header to form an IP datagram.

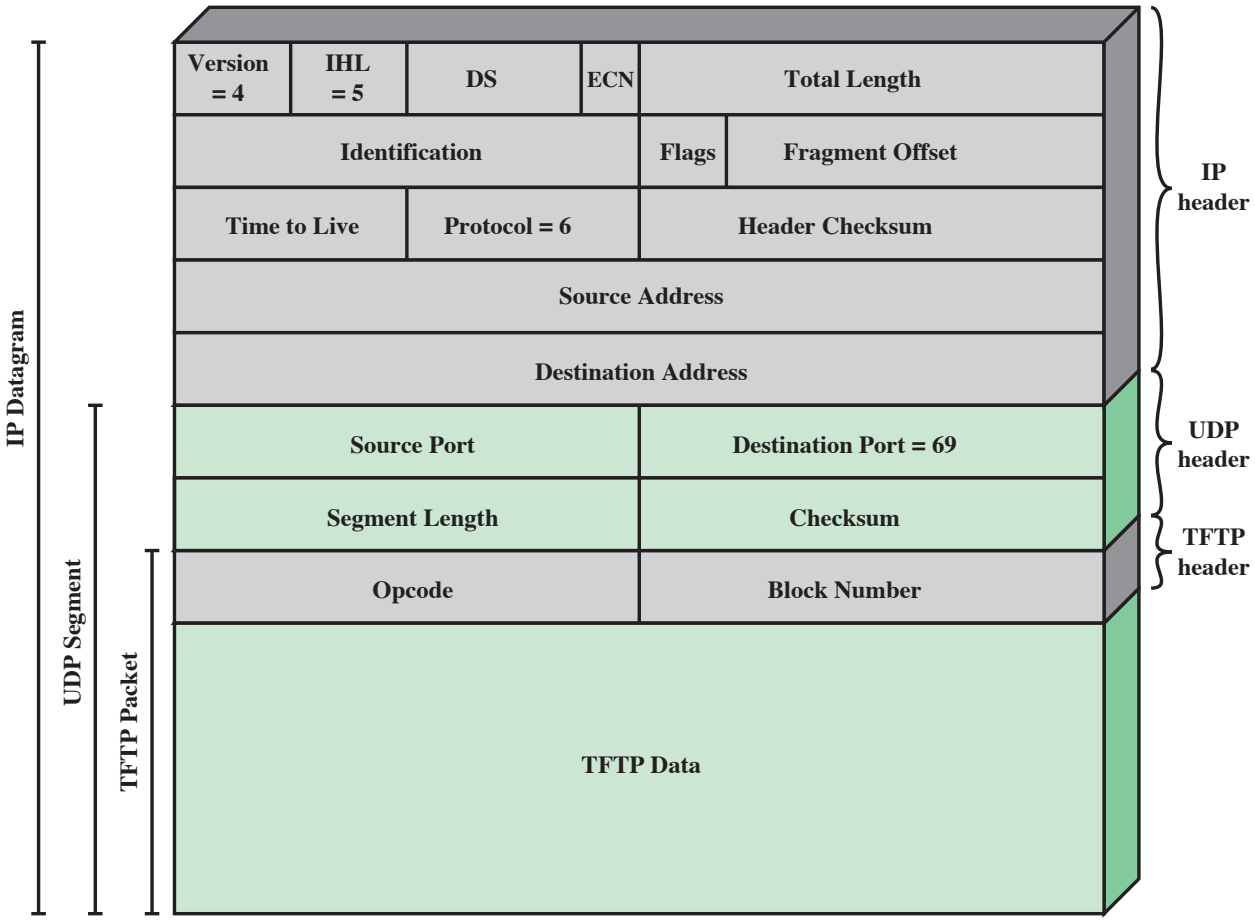

Figure 17.10 A TFTP Packet in Context

Overview of a Transfer

The example illustrated in Figure 17.11 is of a simple file transfer operation from A to B. No errors occur and the details of the option specification are not explored.

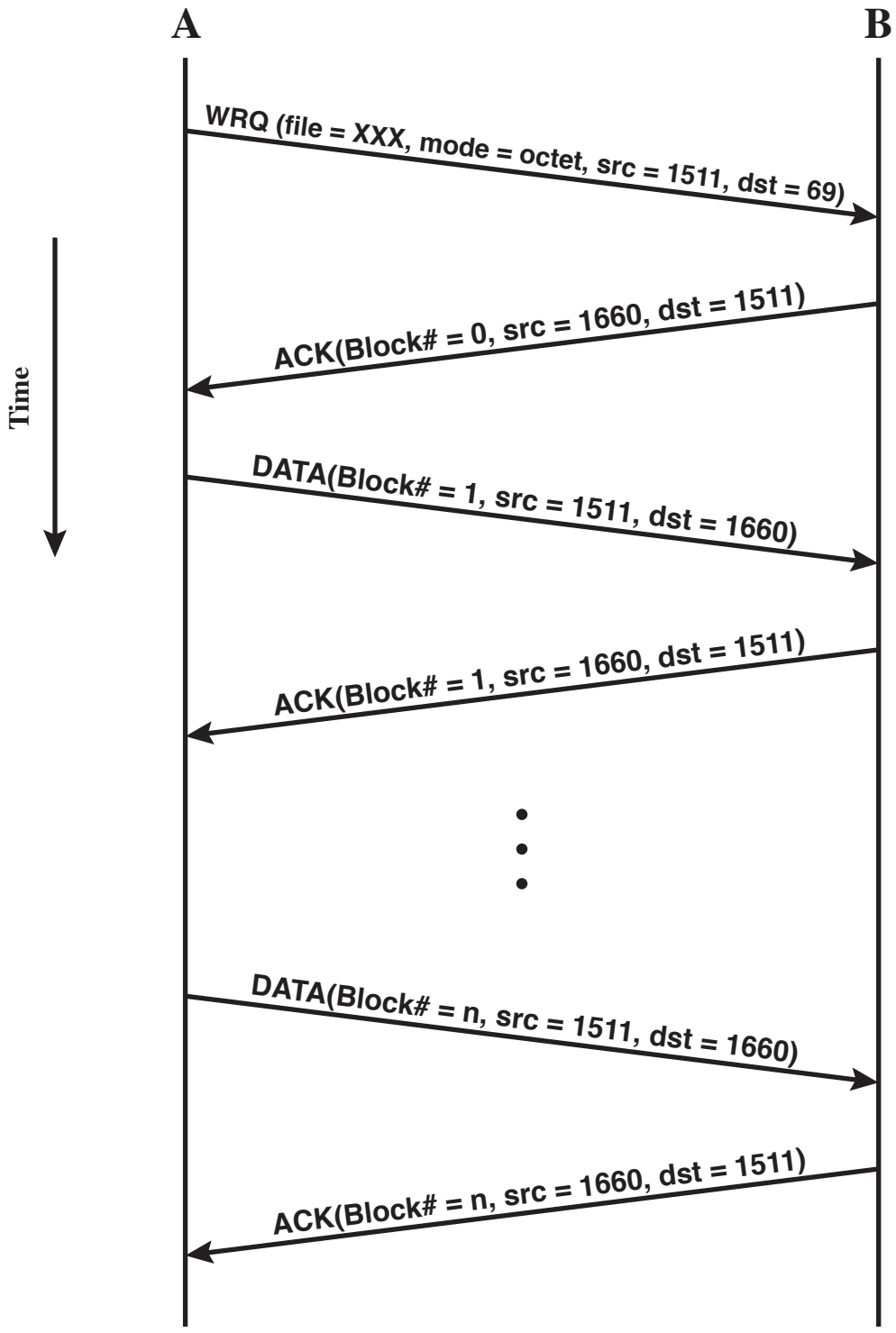

Figure 17.11 Example TFTP Operation

The operation begins when the TFTP module in system A sends a write request (WRQ) to the TFTP module in system B. The WRQ packet is carried as the body of a UDP segment. The write request includes the name of the file (in this case, XXX) and a mode of octet, or raw data. In the UDP header, the destination port number is 69, which alerts the receiving UDP entity that this message is intended for the TFTP application. The source port number is a TID selected by A, in this case 1511. System B is prepared to accept the file and so responds with an ACK with a block number of 0. In the UDP header, the destination port is 1511, which enables the UDP entity at A to route the incoming packet to the TFTP module, which can match this TID with the TID in the WRQ. The source port is a TID selected by B for this file transfer, in this case 1660.

Following this initial exchange, the file transfer proceeds. The transfer consists of one or more data packets from A, each of which is acknowledged by B. The final data packet contains less than 512 bytes of data, which signals the end of the transfer.

Errors and Delays

If TFTP operates over a network or the Internet (as opposed to a direct data link), it is possible for packets to be lost. Because TFTP operates over UDP, which does not provide a reliable delivery service, there needs to be some mechanism in TFTP to deal with lost packets. TFTP uses the common technique of a time-out mechanism. Suppose that A sends a packet to B that requires an acknowledgment (i.e., any packet other than duplicate ACKs and those used for termination). When A has transmitted the packet, it starts a timer. If the timer expires before the acknowledgment is received from B, A retransmits the same packet. If in fact the original packet was lost, then the retransmission will be the first copy of this packet received by B. If the original packet was not lost but the acknowledgment from B was lost, then B

will receive two copies of the same packet from A and simply acknowledges both copies. Because of the use of block numbers, this causes no confusion. The only exception to this rule is for duplicate ACK packets. The second ACK is ignored.

Syntax, Semantics, and Timing

In Section 17.1, it was mentioned that the key features of a protocol can be classified as syntax, semantics, and timing. These categories are easily seen in TFTP. The formats of the various TFTP packets determine the **syntax** of the protocol. The **semantics** of the protocol are shown in the definitions of each of the packet types and the error codes. Finally, the sequence in which packets are exchanged, the use of block numbers, and the use of timers are all aspects of the **timing** of TFTP.