a truck or plane. It sounds like a very complicated way to deliver one message, but this system makes the overall task of delivering many messages easier, not harder. For example, there now can be facilities that only deal with mailbags and do not worry about an individual letter's language or the transportation details.

THE TCP/IP PROTOCOL SUITE

The protocol stack used on the Internet is the Internet Protocol Suite. It is usually called TCP/IP after two of its most prominent protocols, but there are other protocols as well. The *TCP/IP model* is based on a five-layer model for networking. From bottom (the link) to top (the user application), these are the physical, data link, network, transport, and application layers. Not all layers are completely defined by the model, so these layers are "filled in" by external standards and protocols. The layers have names but no numbers, and although sometimes people speak of "Layer 2" or "Layer 3," these are not TCP/IP terms. Terms like these are actually from the OSI Reference Model.

The TCP/IP stack is *open*, which means that there are no "secrets" as to how it works. (There are "open systems" too, but with TCP/IP, the systems do not have to be "open" and often are not.) Two compatible end-system applications can communicate regardless of their underlying architectures, although the connections between layers are not defined.

The OSI Reference Model

The TCP/IP or Internet model is not the only standard way to build a protocol suite or stack. The Open Standard Interconnection (OSI) reference model is a seven-layer model that loosely maps into the five layers of TCP/IP. Until the Web became widely popular in the 1990s, the OSI reference model, with distinctive names and numbers for its layers, was proposed as the standard model for all communication networks. Today, the OSI reference model (OSI-RM) is often used as a learning tool to introduce the functions of TCP/IP.

The TCP/IP stack is comprised of modules. Each module provides a specific function, but the modules are fairly independent. The TCP/IP layers contain relatively independent protocols that can be used depending on the needs of the system to provide whatever function is desired. In TCP/IP, each higher layer protocol is supported by lower layer protocols. The whole collection of protocols forms a type of hourglass shape, with IP in the middle, and more and more protocols up or down from there.

Suite, Stack, and Model

The term "protocol stack" is often used synonymously with "protocol suite" as an implementation of a reference model. However, the term "protocol suite" properly refers to a collection of all the protocols that can make up a layer in the reference model. The Internet protocol suite is an example of the Internet or TCP/IP reference model protocols, and a TCP/IP protocol stack implements one or more of these protocols at each layer.

The TCP/IP Layers

The TCP/IP protocol stack models a series of protocol layers for networks and systems that allows communications between any types of devices. The model consists of five separate but related layers, as shown in Figure 1.9. The Internet protocol suite is based on these five layers. TCP/IP says most about the network and transport layers, and a lot about the application layer. TCP/IP also defines how to interface the network layer with the data link and physical layers, but is not directly concerned with these two layers themselves.

The Internet protocol suite assumes that a layer is there and available, so TCP/IP does not define the layers themselves. The stack consist of protocols, not implementations, so describing a layer or protocols says almost nothing about how these things should actually be built.

Not all systems on a network need to implement all five layers of TCP/IP. Devices using the TCP/IP protocol stack fall into two general categories: a *host* or *end system* (ES) and an *intermediate node* (often a router) or an *intermediate system* (IS). The

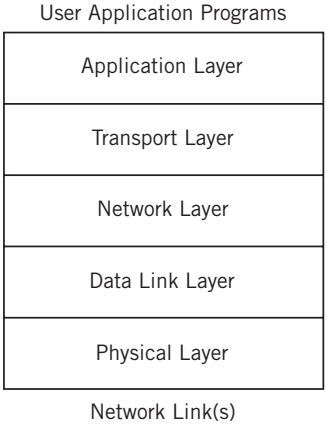


FIGURE 1.9

The five layers of TCP/IP. Older models often show only four layers, combining the physical and data link layers.

intermediate nodes usually only involve the first three layers of TCP/IP (although many of them still have all five layers for other reasons, as we have seen).

In TCP/IP, as with most layered protocols, the most fundamental elements of the process of sending and receiving data are collected into the groups that become the layers. Each layer's major functions are distinct from all the others, but layers can be combined for performance reasons. Each implemented layer has an *interface* with the layers above and below it (except for the application and physical layers, of course) and provides its defined service to the layer above and obtains services from the layer below. In other words, there is a *service interface* between each layer, but these are not standardized and vary widely by operating system.

TCP/IP is designed to be comprehensive and flexible. It can be extended to meet new requirements, and has been. Individual layers can be combined for implementation purposes, as long as the service interfaces to the layers remain intact. Layers can even be split when necessary, and new service interfaces defined. Services are provided to the layer above after the higher layer provides the lower layer with the command, data, and necessary parameters for the lower layer to carry out the task.

Layers on the same system provide and obtain services to and from adjacent layers. However, a *peer-to-peer protocol process* allows the same layers on different systems to communicate. The term *peer* means every implementation of some layer is essentially equal to all others. There is no "master" system at the protocol level. Communications between peer layers on different systems use the defined protocols appropriate to the given layer.

In other words, *services* refer to communications between layers within the same process, and *protocols* refer to communications between processes. This can be confusing, so more information about these points is a good idea.

Protocols and Interfaces

It is important to note that when the layers of TCP/IP are on different systems, they are *only* connected at the physical layer. Direct peer-to-peer communication between all other layers is impossible. This means that all data from an application have to flow "down" through all five layers at the sender, and "up" all five layers at the receiver to reach the correct process on the other system. These data are sometimes called a *service data unit* (SDU).

Each layer on the sending system adds information to the data it receives from the layer above and passes it all to the layer below (except for the physical layer, which has no lower layers to rely on in the model and actually has to send the bits in a form appropriate for the communications link used).

Likewise, each layer on the receiving system unwraps the received message, often called a *protocol data unit* (PDU), with each layer examining, using, and stripping off the information it needs to complete its task, and passing the remainder up to the next layer (except for the application layer, which passes what's left off to the application program itself). For example, the data link layer removes the wrapper meant for it, uses it to decide what it should do with this data unit, and then passes the remainder up to the network layer.

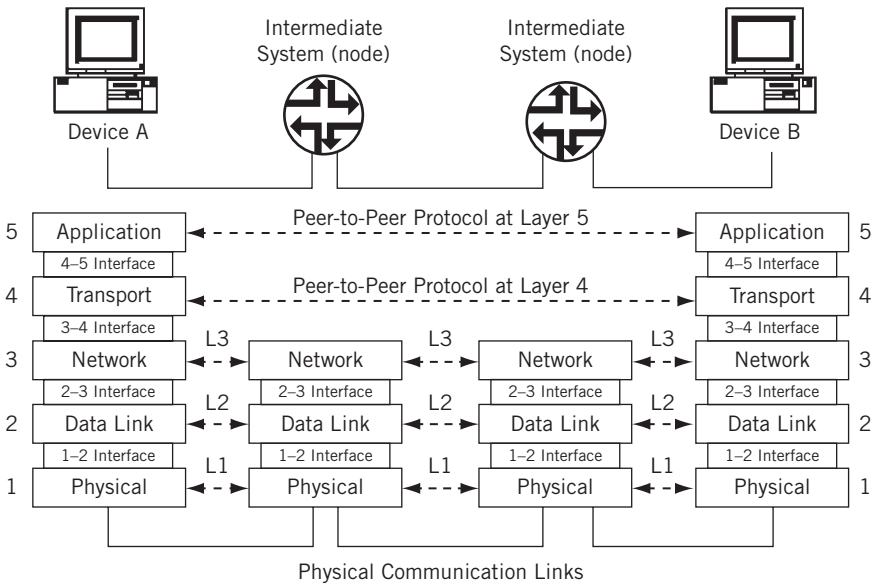


FIGURE 1.10

Protocols and interfaces, showing how devices are only physically connected at the lowest layer (Layer 1). Note that functionally, intermediate nodes only require the bottom three layers of the model.

The whole interface and protocol process is shown in Figure 1.10. Although TCP/IP layers only have names, layer numbers are also used in the figure, but only for illustration. (The numbers come from the ISO-RM.)

As shown in the figure, there is a natural grouping of the five-layer protocol stack at the network layer and the transport layer. The lower three layers of TCP/IP, sometimes called the network support layers, must be present and functional on all systems, regardless of the end system or intermediate node role. The transport layer links the upper and lower layers together. This layer can be used to make sure that what was sent was received, and what was sent is useful to the receiver (and not, for example, a stray PDU misdirected to the host or unreasonably delayed).

The process of encapsulation makes the whole architecture workable. Encapsulation of one layer's information inside another layer is a key part of how TCP/IP works.

Encapsulation

Each layer uses encapsulation to add the information its peer needs on the receiving system. The network layer adds a header to the information it receives from the transport at the sender and passes the whole unit down to the data link layer. At the receiver,

the network layer looks at the control information, usually in a *header*, in the data it receives from the data link layer and passes the remainder up to the transport layer for further processing. This is called encapsulation because one layer has no idea what the structure or meaning of the PDU is at other layers. The PDU has several more or less official names for the structure at each layer.

The exception to this general rule is the data link layer, which adds both a *header* and a *trailer* to the data it receives from the network layer. The general flow of encapsulation in TCP/IP is shown in Figure 1.11. Note that on the transmission media itself (or communications link), there are only bits, and that some "extra" bits are added by the communication link for its own purposes. Each PDU at the other layers is labeled as data for its layer, and the headers are abbreviated by layer name. The exception is the second layer, the data link layer, which shows a header and trailer added at that level of encapsulation.

Although the intermediate nodes are not shown, these network devices will only process the data (at most) through the first three layers. In other words, there is no transport layer to which to pass network-layer PDUs on these systems for data communications (management is another issue).

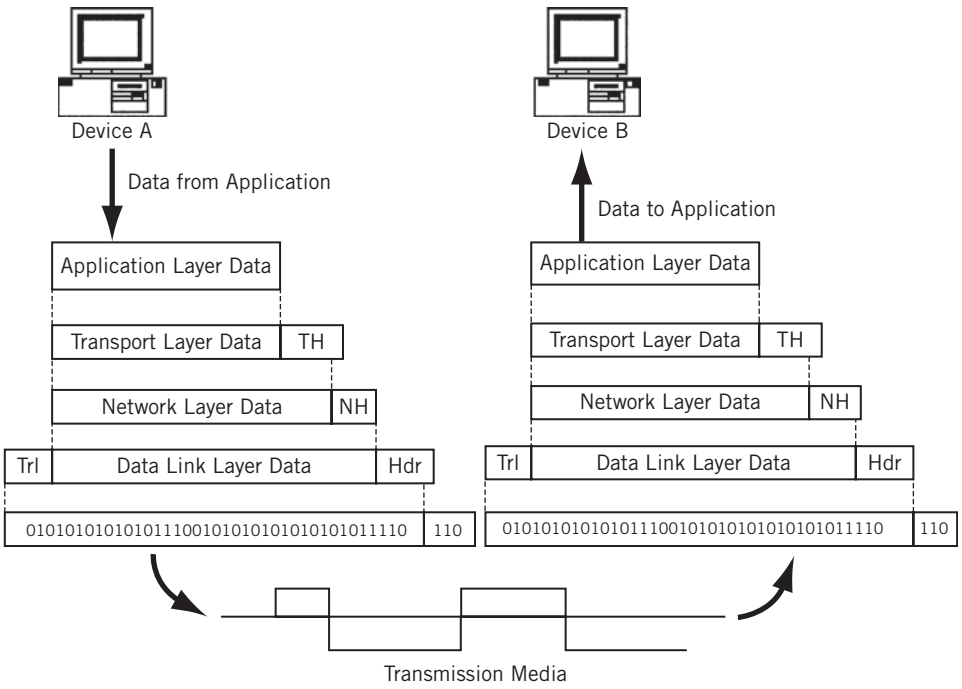


FIGURE 1.11

TCP/IP encapsulation and headers. The unstructured stream of bits represents frames with distinct content.

THE LAYERS OF TCP/IP

TCP/IP is mature and stable, and is the only protocol stack used on the Internet. This book is all about networking with TCP/IP, but it is easy to get lost in the particulars of TCP/IP if some discussion of the general tasks that TCP/IP is intended to accomplish is not included. This section takes a closer look at the TCP/IP layers, but only as a general guide to how the layers work.

TCP/IP Layers in Brief

- **Physical Layer:** Contains all the functions needed to carry the bit stream over a physical medium to another system.
- **Data Link Layer:** Organizes the bit stream into a data unit called a "frame" and delivers the frame to an adjacent system.
- **Network Layer:** Delivers data in the form of a packet from source to destination, across as many links as necessary, to non-adjacent systems.
- **Transport Layer:** Concerned with process-to-process delivery of information.
- **Application Layer:** Concerned with differences in internal representation, user interfaces, and anything else that the user requires.

The Physical Layer

The physical layer contains all the functions needed to carry the bit stream over a physical medium to another system. Figure 1.12 shows the position of the physical layer to the data link layer and the transmission medium. The transmission medium forms a pure "bit pipe" and should not change the bits sent in any way. Now, transmission "on the wire" might send bits through an extremely complex transform, but the goal is to enable the receiver to reconstruct the bit stream exactly as sent. Some information in the form of *transmission framing* can be added to the data link layer data, but this is only used by the physical layer and the transmission medium itself. In some cases, the transmission medium sends a constant idle bit pattern until interrupted by data.

Physical layer specifications have four parts: mechanical, electrical or optical, functional, and procedural. The mechanical part specifies the physical size and shape of the connector itself so that components will plug into each other easily. The electrical/optical specification determines what value of voltage or line condition determines whether a pin is active or what exactly represents a 0 or 1 bit. The functional specification specifies the function of each pin or lead on the connector (first lead is send, second is receive, and so on). The procedural specification details the sequence of actions that must take place to send or receive bits on the interface. (For Ethernet, the send pair is activated, then a "preamble" is sent, and so forth.) The Ethernet twisted-pair interfaces from the IEEE are common implementations of the physical layer that includes all these elements.

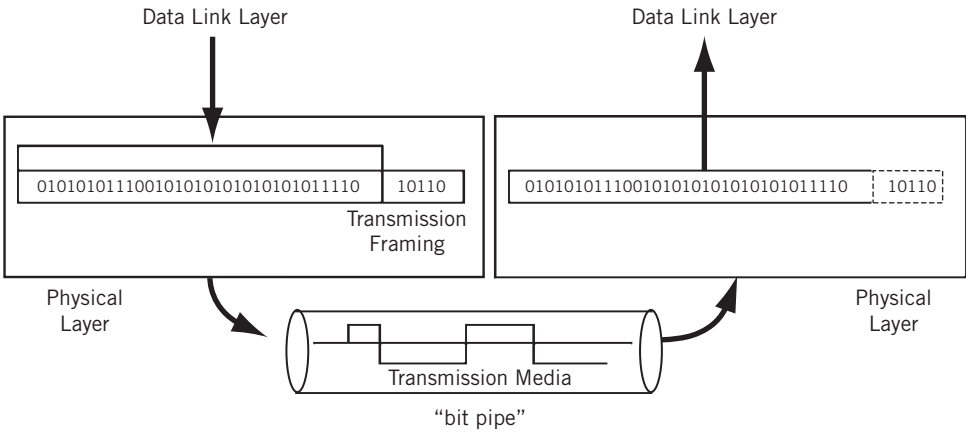


FIGURE 1.12

The physical layer. The transmission framing bits are used for transmission media purposes only, such as low-level control.

There are other things that the physical layer must determine, or be configured to expect.

Data rate—This transmission rate is the number of bits per second that can be sent. It also defines the duration of a symbol on the wire. Symbols usually represent one or more bits, although there are schemes in which one bit is represented by multiple symbols.

Bit synchronization—The sender and receiver must be *synchronized* at the symbol level so that the number of bits expected per unit time is the same. In other words, the sender and receiver *clocks* must be synchronized (timing is in the millisecond or microsecond range). On modern links, the timing information is often "recovered" from the received data stream.

Configuration—So far we've assumed simple *point-to-point links*, but this is not the only way that systems are connected. In a *multipoint configuration*, a link connects more than two devices, and in a multisystem *bus/broadcast* topology such as a LAN, the number of systems can be very high.

Topology—The devices can be arranged in a number of ways. In a full *mesh* topology, all devices are directly connected and *one hop* away, but this requires a staggering amount of links for even a modest network. Systems can also be arranged as a *star* topology, with all systems reachable through a central system. There is also the *bus* (all devices are on a common link) and the *ring* (devices are chained together, and the last is linked to the first, forming a ring).

Mode—So far, we've only talked about one of the systems as the sender and the other as the receiver. This is operation in *simplex mode*, where a device can only send or receive, such as with weather sensors reporting to a remote

weather station. More realistic devices use *duplex mode*, where all systems can send or receive with equal facility. This is often further distinguished as *half-duplex* (the system can send and receive, but not at the same time) and *full-duplex* (simultaneous sending and receiving).

The Data Link Layer

Bits are just bits. With only a physical layer, System A has no way to tell System B, "Get ready some bits," "Here are the bits," and "Did you get those bits okay?" The data link layer solves this problem by organizing the bit stream into a data unit called a frame.

It is important to note that frames are the data link layer PDUs, and these are not the same as the physical layer transmission frames mentioned in the previous section. For example, network engineers often speak about *T1 frames* or *SONET frames*, but these are distinct from the data link layer frames that are carried inside the T1 or SONET frames. Transmission frames have control information used to manage the physical link itself and has little to do directly with process-to-process communications. This "double-frame" arrangement might sound redundant, but many transmission frames originated with voice because digitized voice has no framing at the "data link" layer.

The data link layer moves bits across the link and can add reliability to the raw communications link. The data link layer can be very simple, or make the link appear error-free to the layer above, the network layer. The data link layer usually adds both a header and trailer to the data presented by the network layer. This is shown in Figure 1.13.

The frame header typically contains a source and destination address (known as the "physical address" since it refers to the physical communication port) and some control information. The control information is data passed from one data link layer to the

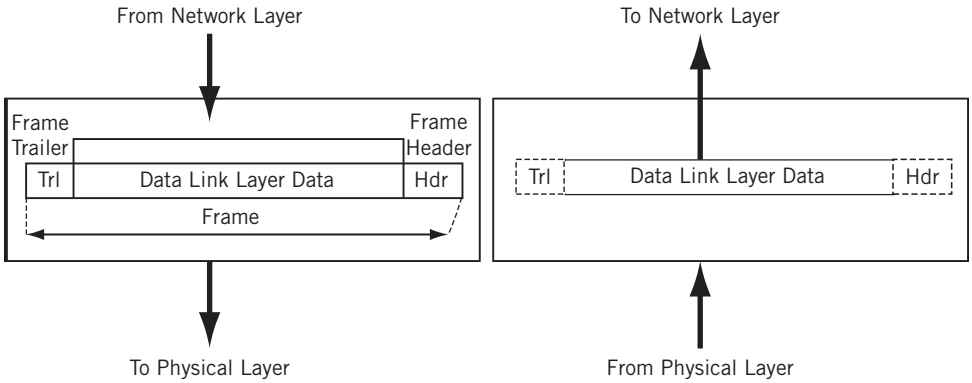


FIGURE 1.13

The data link layer, showing that data link layer frames have both header and trailer.

other data link layer, and not user data. The body of the frame contains the sequence of bits being transferred across the network. The trailer usually contains information used in detecting bit errors (such as cyclical redundancy check [CRC]). A maximum size is associated with the frame that cannot be exceeded because all systems must allocate memory space (buffers) for the data. In a networking context, a buffer is just special memory allocated for communications.

The data link layer performs framing, physical addressing, and error detection (error *correction* is another matter entirely, and can be handled in many ways, such as by resending a copy of the frame that had the errors). However, when it comes to frame error detection and correction in the real world, error detection bits are sometimes ignored and frames that defy processing due to errors are simply discarded. This does not mean that error detection and correction are not part of the data link layer standards: It means that in these cases, ignoring and discarding are the chosen methods of implementation. In discard cases, the chore of handling the error condition is "pushed up the stack" to a higher layer protocol.

This layer also performs *access control* (this determines whose turn it is to send over or control the link, an issue that becomes more and more interesting as the number of devices sharing the link grows). In LANs, this *media access control* (MAC) forms a sublayer of the data link layer and has its own addressing scheme known (not surprisingly) as the MAC layer address or *MAC address*. We'll look at MAC addresses in the next chapter. For now, it is enough to note that LANs such as Ethernet do not have "real" physical layer addresses and that the MAC address performs this addressing function.

In addition, the data link layer can perform some type of *flow control*. Flow control makes sure senders do not overwhelm receivers: a receiver must have adequate time to process the data arriving in its buffers. At this layer, the flow control, if provided, is link-by-link. (We'll see shortly that end-to-end—host-to-host—flow control is provided by the transport layer.) LANs do not usually provide flow control at the data link layer, although they can.

Not all destination systems are directly reachable by the sender. This means that when bits at the data link layer are sent from an originating system, the bits do not arrive at the destination system as the "next hop" along the way. Directly reachable systems are called *adjacent systems*, and adjacent systems are always "one hop away" from the sender. When the destination system is not directly reachable by the sender, one or more intermediate nodes are needed. Consider the network shown in Figure 1.14.

Now the sender (System A) is not directly connected to the receiver (System B). Another system, System 3, receives the frame and must forward it toward the destination. This system is usually called a *switch* or *router* (there are even other names), depending on internal architecture and network role. On a WAN (but not on a LAN), this second frame is a different frame because there is no guarantee that the second link is identical to the first. Different links need different frames. Identical frames are only delivered to systems that are directly reachable, or adjacent, to the sender, such as by an Ethernet switch on a LAN.

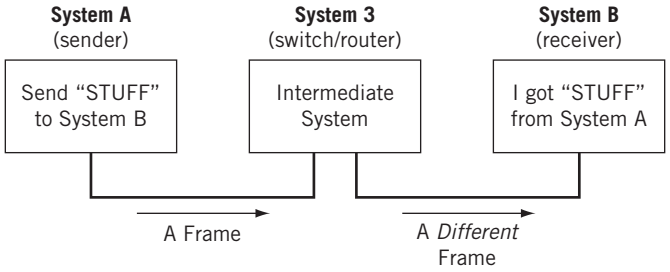


FIGURE 1.14

A more complex network. Note that the frames are technically different even if the same medium is used on both links.

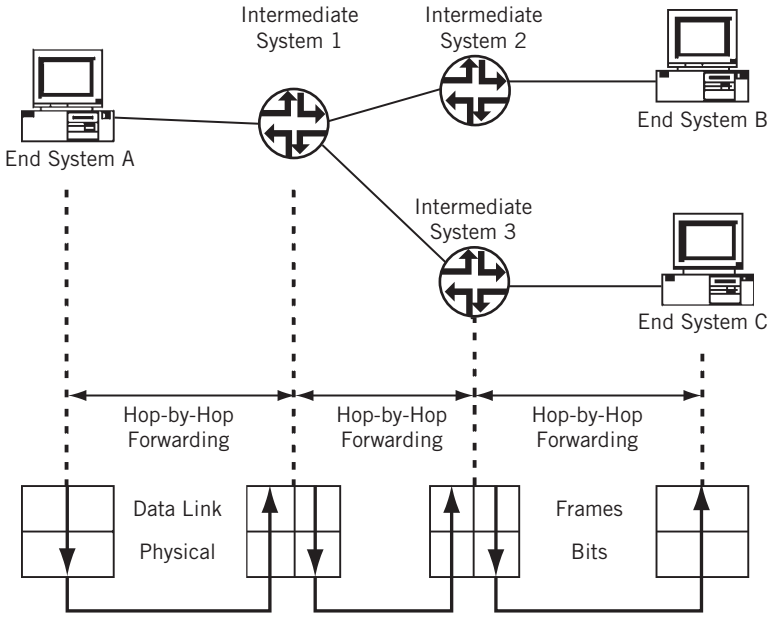


FIGURE 1.15

Hop-by-hop forwarding of frames. The intermediate systems also have a Layer 3, but this is not shown in the figure for clarity.

Networking with intermediate systems is called *hop-by-hop* delivery. A "hop" is the usual term used on the Internet or a router network to indicate the forwarding of a packet between one router or another (or between a host and router). Frames can "hop" between Layer 2 switches, but the term is most commonly used for Layer 3 router hops (which can consist of multiple switch-to-switch frame "hops"). There can be more than one intermediate system between the source and destination end systems, of course, as shown in Figure 1.15. Consider the case where End System A is sending a bit stream to End System C.

Note that the intermediate systems (routers) have *two* distinct physical and data link layers, reflecting the fact that the systems have two (and often more) communication links, which can differ in many ways. (The figure shows a typical WAN configuration with point-to-point links, but routers on LANs, and on some types of public data service WANs, can be deployed in more complicated ways.)

However, there is something obviously missing from this figure. There is no connection between the data link layers on the intermediate systems! How does the router know to which output port and link to forward the data in order to ultimately reach the destination? (In the figure, note that Intermediate System 1 can send data to either Intermediate System 2 or Intermediate System 3, but only through Intermediate System 3, which forwards the data, is the destination reachable.)

These forwarding decisions are made at the TCP/IP network layer.

The Network Layer

The network layer delivers data in the form of a *packet* from source to destination, across as many links as necessary. The biggest difference between the network layer and the data link layer is that the data link layer is in charge of data delivery between *adjacent* systems (directly connected systems one hop away), while the network layer delivers data to systems that are not directly connected to the source. There can be many different types of data link and physical layers on the network, depending on the variety of the link types, but the network layer is essentially the same on all systems, end systems, and intermediate systems alike.

Figure 1.16 shows the relationship between the network layer and the transport layer above and the data link layer below. A packet header is put in place at the sender and interpreted by the receiver. A router simply looks at the packet header and makes a forwarding decision based on this information. The transport layer does not play a role in the forwarding decision.

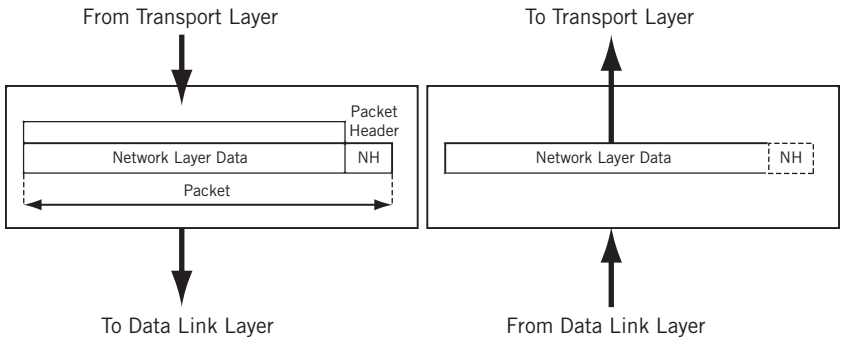


FIGURE 1.16

The network layer. These data units are packets with their own destination and source address formats.

How does the network layer know where the packet came from (so the sender can reply)? The key concept at the network layer is the *network address*, which provides this information. In TCP/IP, the network address is the *IP address*.

Every system in the network receives a network address, whether an end system or intermediate system. Systems require at least one network address (and sometimes many more). It is important to realize that this network address is different from, and independent of, the physical address used by the frames that carry the packets between adjacent systems.

Why should the systems need two addresses for the two layers? Why can't they just both use either the data link ("physical") address or the network address at *both* layers? There are actually several reasons. First, LAN addresses like those used in Ethernet come from one group (the IEEE), while those used in TCP/IP come from another group (ICANN). Also, the IP address is universally used on the Internet, while there are many types of physical addresses. Finally, there is no systematic assignment of physical addresses (and many addresses on WANs can be duplicates and so have "local significance only"). On the other hand, IP network addresses are globally administered, unique, and have a portion under which many devices are grouped. Therefore, many devices can be addressed concisely by this *network portion* of the IP address.

A key issue is how the network addresses "map" to physical addresses, a process known generally as *address resolution*. In TCP/IP, a special family of address resolution protocols takes care of this process.

The network address is a logical address. Network addresses should be organized so that devices can be grouped under a part of that address. In other words, the network address should be organized in a fashion similar to a telephone number, for example, 212-555-1212 in the North American public switched telephone network (PSTN). The sender need only look at the area code or "network" portion of this address (212) to determine if the destination is local (area codes are the same) or needs to be sent to an intermediate system to reach the 212 area code (source and destination area codes differ).

For this scheme to work effectively, however, all telephones that share the 212 area code should be grouped together. The whole telephone number beginning with 212 therefore means "*this* telephone in the 212 area code." In TCP/IP, the network address is the beginning of the device's complete IP address. A group of hosts is gathered under the network portion of the IP address. IP network addresses, like area codes, are globally administered to prevent duplication, while the rest of the IP address, like the rest of the telephone number, is locally administered, often independently.

In some cases, the packet that arrives at an intermediate system inside a frame is too large to fit inside the frame that must be sent out. This is not uncommon: different link and LAN types have different maximum frame sizes. The network layer must be able to *fragment* a data unit across multiple frames and reassemble the fragments at the destination. We'll say more about *fragmentation* in a later chapter.

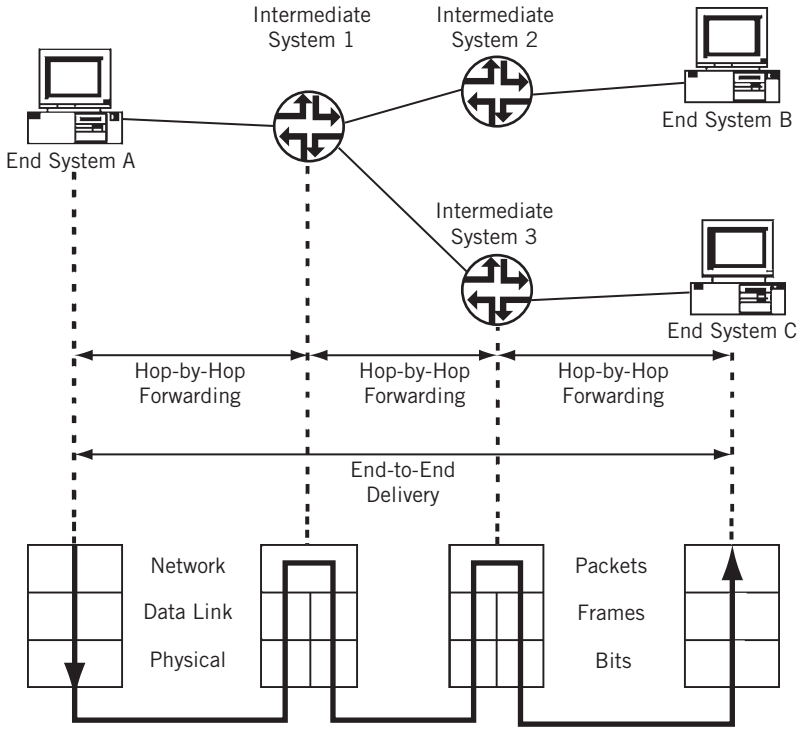


FIGURE 1.17

Source-to-destination delivery at the network layer. The intermediate systems now have all three required layers.

The network layer uses one or more *routing tables* to store information about reachable systems. The routing tables must be created, maintained, and purged of old information as the network changes due to failures, the addition or deletion of systems and links, or other configuration changes. This whole process of building tables to pass data from source to destination is called *routing*, and the use of these tables for packet delivery is called *forwarding*. The forwarding of packets inside frames always takes place hop by hop. This is shown in Figure 1.17, which adds the network layer to the data link layers already present and distinguishes between hop-by-hop forwarding and end-to-end delivery.

On the Internet, the intermediate systems that act at the packet level (Layer 3) are called *routers*. Devices that act on frames (Layer 2) are called *switches*, and some older telephony-based WAN architectures use switches as intermediate network nodes. Whether a node is called a switch or router depends on how they function internally.

In a very real sense, the network layer is at the very heart of any protocol stack, and TCP/IP is no exception. The protocol at this layer is IP, either IPv4 or IPv6 (some think that IPv6 is distinct enough to be known as TCPv6/IPv6).

The Transport Layer

Process-to-process delivery is the task of the transport layer. Getting a packet to the destination system is not quite the same thing as determining which process should receive the packet's *content*. A system can be running file transfer, email, and other network processes all at the same time, and all over a single physical interface. Naturally, the destination process has to know on which process the sender originated the bits inside the packet in order to reply. Also, systems cannot simply transfer a huge multimegabit file all in one packet. Many data units exceed the maximum allowable size of a packet.

This process of dividing message content into packets is known as *segmentation*. The network layer forwards each and every packet independently, and does not recognize any relationship between the packets. (Is this a file transfer or email packet? The network layer does not care.) The transport layer, in contrast, can make sure the whole *message*, often strung out in a sequence of packets, arrives in order (packets can be delivered out of sequence) and intact (there are no errors in the entire message). This function of the transport layer involves some method of flow control and error control (error detection *and* error correction) at the transport layer, functions which are absent at the network layer. The transport-layer protocol that performs all of these functions is TCP.

The transport-layer protocol does not have to do any of this, of course. In many cases, the content of the packet forms a complete unit all by itself, called a *datagram*. (The term "datagram" is often used to refer to the whole IP packet, but not in this book.) Self-contained datagrams are not concerned with sequencing or flow control, and these functions are absent in the User Datagram Protocol (UDP) at the transport layer.

So there are two very popular protocol packages at the transport layer:

- *TCP*—This is a connection-oriented, "reliable" service that provides ordered delivery of packet contents.
- *UDP*—This is a connectionless, "unreliable" service that does *not* provide ordered delivery of packet contents.

In addition to UDP and TCP, there are other transport-layer protocols that can be used in TCP/IP, all of which differ in terms of how they handle transport-layer tasks. Developers are not limited to the standard choices for applications. If neither TCP nor UDP nor any other defined transport-layer service is appropriate for your application, you can write your own transport-layer protocols and get others to adapt it (or use your application package exclusively).

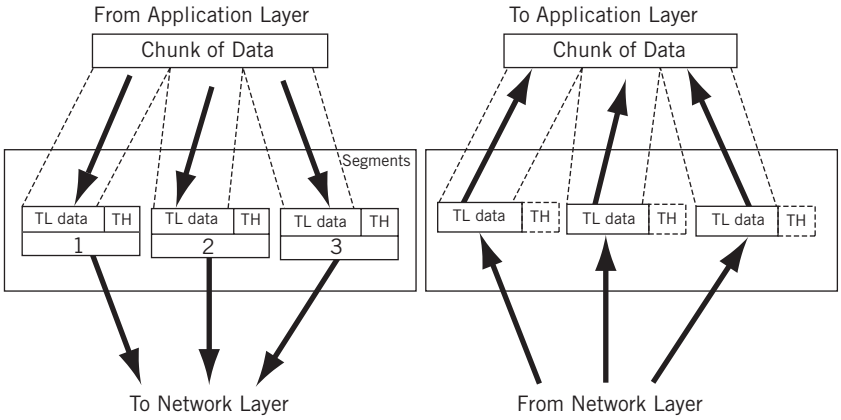


FIGURE 1.18

The transport layer, showing how data are broken up if necessary and reassembled at the destination.

In TCP/IP, it is often said that the network layer (IP itself) offers an "unreliable" or "best effort" service, while the transport layer adds "reliability" in the form of flow and error control. Later in this book, we'll see why these terms are unfortunate and what they really mean.

The network layer gets a single packet to the right system, and the transport layer gets the entire message to the right process. Figure 1.18 shows the transport layer breaking up a message at the sender into three pieces (each labeled "TL data" for transport-layer data and "TH" for transport-layer header). The figure then shows the transport layer reassembling the message at the receiver from the various *segments* that make up a message. In TCP/IP, there are also data units known as *datagrams*, which are always handled as self-contained units. There are profound differences between how the transport layer treats segments and datagrams, but this figure is just a general illustration of segment handling.

The functions that the transport layer, which in some protocols is called the end-to-end layer, might have to include follow:

Process addressing and multiplexing—Also known as "service-point addressing," the transport layer has to decide which process originated the message and to which process the message must be delivered. These are also known as *port addresses* in TCP/IP. Port addresses are an important portion of the application *socket* in TCP/IP.

Segment handling—In cases where each message is divided into segments, each segment has a sequence number used to put the message back together at the destination. When datagrams are used, each data unit is handled independently and sequencing is not necessary.

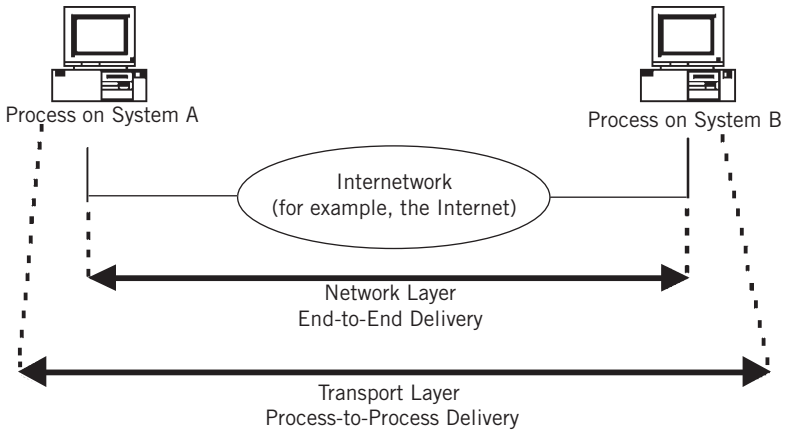


FIGURE 1.19

Reliable process-to-process delivery with the transport layer.

Connection control—The transport layer can be *connectionless* or *connection-oriented* (in fact, several layers can operate in either one of these ways). Connectionless (CL) layers treat every data unit as a self-contained, independent unit. Connection-oriented (CO) layers go through a three-phase process every time there is data to send to a destination after an idle period (connection durations can vary). First, some control messages establish the connection, then the data are sent (and exchanged if replies are necessary), and finally the connection is closed. Many times, a comparison is made between a telephone conversation ("dial, talk, hang up") with connections and an intercom ("push and talk any time") for connectionless communications, but this is not precise. Generally, segments are connection-oriented data units, and datagrams are connectionless data units.

Flow control—Just as with the data link layer, the transport layer can include flow control mechanisms to prevent senders from overwhelming receivers. In this case, however, the flow control is end-to-end rather than link-by-link. Datagrams do not require this service.

Error control—This is another function that can be performed at the data link layer, but again end-to-end at the transport layer rather than link-by-link. Communications links are not the only source of errors, which can occur inside a system as well. Again, datagrams do not require this service.

Figure 1.19 shows the relationship between the network layer and transport layer more clearly. The network layer operates from network interface to network interface, while the transport layer is more specific and operates from process to process.

The Application Layer

It might seem that once data are transferred from end-system process to end-system process, the networking task is pretty much complete. There is a lot that still needs to be done at the application level itself. In models of protocol stacks, it is common to place another layer between the transport layer and the user, the application layer. However, the TCP/IP protocol stack really stops at the transport layer (where TCP and UDP are). It is up to the application programmer to decide what should happen at the client and server level at that point, although there are individual RFCs for guidance, such as for FTP.

Although it is common to gather these TCP/IP applications into their own layer, there really is no such thing in TCP/IP as an application layer to act as some kind of "glue" between the application's user and the network.

In nearly all TCP/IP stacks, the application layer is part of the application process. In spite of the lack of a defined layer, a TCP/IP application might still have a lot to do, and in some ways the application layer is the most complex "layer" of all.

There are two major tasks that the application often needs to accomplish: session support and conversion of internal representation. Not all applications need both, of course, and some applications might not need either, but this overview includes both major functions.

Session Support

A session is a type of *dialog controller* between two processes that establishes, maintains, and synchronizes (controls) the interaction (dialog). A session decides if the communication can be half-duplex (both ends take turns sending) or full-duplex (both ends can send whenever they want). It also keeps a kind of "history" of the interaction between endpoints, so that when things go wrong or when the two communicate again, some information does not have to be resent.

In practical terms, the session consists of all "state variables" necessary to construct the history of the connection between the two devices. It is more difficult, but not impossible, to implement sessions in a connectionless environment because there is no easy way to associate the variables with a convenient label.

Internal Representation Conversion

The role of internal representation conversion is to make sure that the data exchange over the network is useful to the receivers. If the internal representation of data differs on the two systems (integer size, bit order in memory, etc.), the application layer translates between the formats so the application program does not have to. This layer can also provide encryption and compression functions, although it is more common to implement these last two functions separately from the network.

Standard protocol specifications can use the Abstract Syntax Notation 1 (ASN.1) definitions for translation purposes. ASN.1 can be used in programming, network

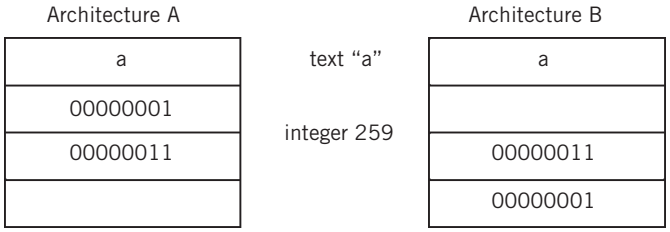


FIGURE 1.20

Internal representation differences. Integers can have different bit lengths and can be stored differently in memory.

management, and other places. ASN.1 defines various things such as which bit is "first on the wire" regardless of how it is stored internally, how many bits are to be sent for the numbers 0 through 255 (8), and so on. Everything can be translated into ASN.1, sent across the network, and translated back to whatever internal format is required at the destination.

The role of internal representation conversion is shown in Figure 1.20. The figure shows four sequential memory locations, each storing the letter "a" followed by the integer 259. Note that not only are there differences between the amount of memory addressed at once, but also in the *order* of the bits for numerics.

In some protocol stacks, the application program can rely on the services of a fully functional conversion for internal representation to perform these services. However, in TCP/IP, every network application program must do these things for itself.

Applications in TCP/IP

TCP/IP does not provide session or presentation services directly to an application. Programmers are on their own, but this does not mean they have to create everything from scratch. For example, applications can use a character-based presentation service called the Network Virtual Terminal (NVT), part of the Internet's telnet remote access specification. Other applications can use Sun's External Data Representation (XDR) or IBM's (and Microsoft's) NetBIOS programming libraries for presentation services. In this respect, there are many presentation layer services that TCP/IP can use, but there is no formal presentation service standard in TCP/IP that all applications must use.

Host TCP/IP implementations typically provide a range of applications that provide users with access to the data handled by the transport-layer protocols. These applications use a number of protocols that are not part of TCP/IP proper, but are used with TCP/IP. These protocols include the Hyper-Text Transfer Protocol (HTTP) used by Web browsers, the Simple Message Transfer Protocol (SMTP) used for email, and many others.

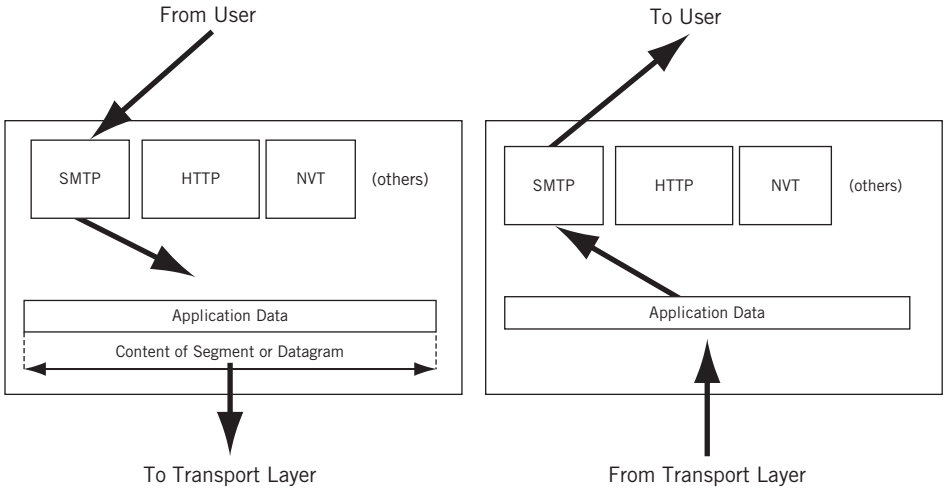


FIGURE 1.21

TCP/IP applications, showing how multiple applications can all share the same network connection.

In TCP/IP, the application protocol, the application service, and the user application itself often share the same name. The file transfer protocol in TCP/IP, called FTP, is at once an application protocol, an application service, and an application run by a user. It can sometimes be confusing as to just which aspect of FTP is under discussion.

The role of TCP/IP applications is shown in Figure 1.21. Note that this "layer" sits on top of the TCP/IP protocol stack and interfaces with programs or users directly.

Some protocols provide separate layers for sessions, internal representation conversion, and application services. In practice, these are seldom implemented independently. It just makes more sense to bundle them together by major application, as in TCP/IP.

THE TCP/IP PROTOCOL SUITE

To sum up, the five layers of TCP/IP are physical, data link, network, transport, and application. The TCP/IP stack is a hierarchical model made up of interactive modules. Each module provides a specific function. In TCP/IP, the layers contain relatively independent protocols that can be "mixed and matched" depending on the needs of the system to provide whatever function is desired. TCP/IP is hierarchical in the sense that each higher layer protocol is supported by one or more lower layer protocols.

Figure 1.22 maps some of the protocols used in TCP/IP to the various layers of TCP/IP. Every protocol in the figure will be discussed in this book, most in chapters all their own.

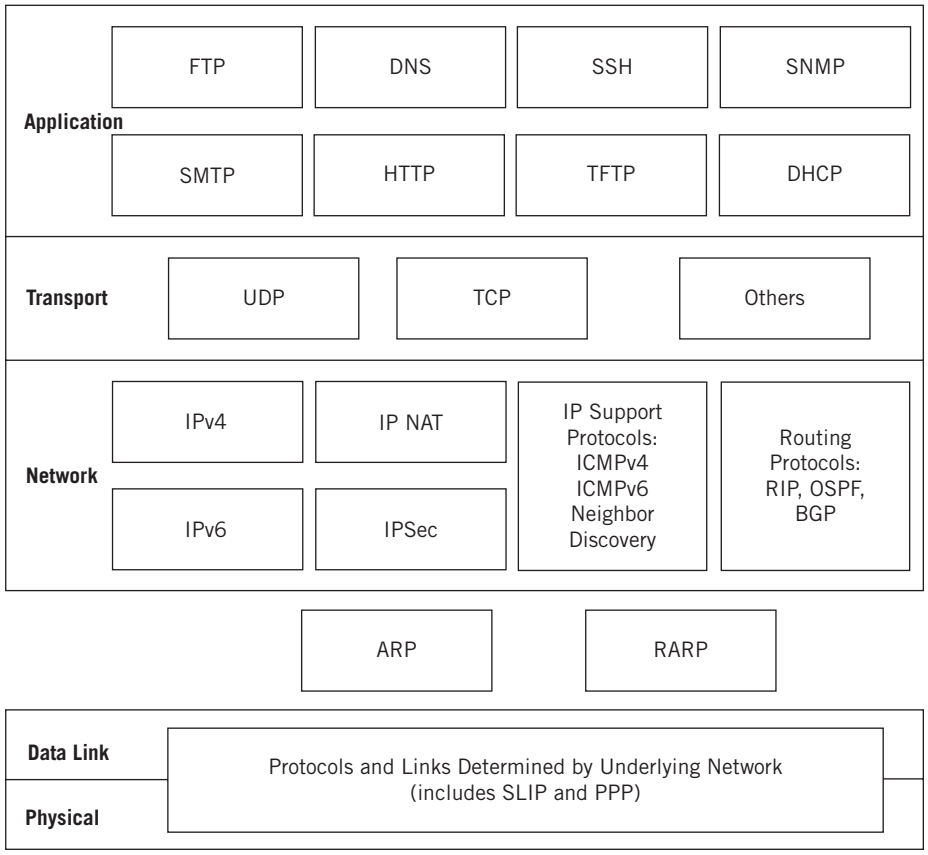


FIGURE 1.22

TCP/IP protocols and layers. Note the position of some protocols between layers.

With few exceptions, the TCP/IP protocol suite does not really define any low-level protocols below the network layer. TCP/IP usually specifies how to put IP packets into frames and how to get them out again. Many RFCs define IP *mapping* into these lower-layer protocols. We'll talk more about this mapping process in Chapter 2.

Table 2.1 Port and Sockets

Value	lnxclient	lnxserver
IP address	10.10.12.166	10.10.11.66
Port	32825	55555
Socket	10.10.12.166:32825	10.10.11.66:55555

Let's look at the fields that are emphasized. First, we have captured an Ethernet II frame with an IPv4 packet inside. The frame's type field value of `0x800` determines this. In the IP packet, the message from the client to the server, which starts on the next line, the source address is `10.10.12.166` (`lnxclient`) and the destination address is `10.10.11.66` (`lnxserver`), as they should be.

We can ignore the rest of the IP header fields for now, and skip down to where the source and destination port are highlighted. The source port chosen by the client is `32825` and the port on the server that will receive the data is `55555`. We decided that `55555` would be the server port, and the client chose a port number to use based on certain rules, which we will talk about in a later chapter.

Now that we know the IP addresses and ports used, we can determine the socket at each host. This is shown in Table 2.1.

THE TCP/IP PROTOCOL STACK

The layering of TCP/IP is important if IP packets are to run on almost any type of network. The IP packet layer is only one layer, and from the TCP/IP perspective, the layer or layers below the IP layer are not as important as the overall flow of packets from one host (end system) to another across the network.

Layering means that you only have to adapt one type of packet to an underlying network type to get the entire TCP/IP suite. Once the packet has been "framed," you need not worry about TCP/UDP, or any other protocol: they come along for the ride with the layering. Only the IP layer has to deal with the underlying hardware.

All that really matters is that the device at the receiving end understands the type of IP packet encapsulation used at the sending end. If only one form of packet encapsulation was used, the IP packets could remain inside the frame with a globally unique MAC address from source to destination. Network nodes could forward the frame without having to deal with the packet inside. We'll talk more about the differences between forwarding frames and forwarding packets later on in this book.

TCP/IP is considered to be a *peer protocol* stack, which means that every implementation of TCP/IP is considered to have the same capabilities as every other. There are no "restricted" or "master" versions of TCP/IP that anyone need be concerned about. So, for example, there is no special server stack needed.

However, this does not mean that all protocol stacks function in precisely the same way. TCP/IP, like many other protocol stacks, is implemented according to a model known as the *client–server model*.

THE CLIENT–SERVER MODEL

The hosts that run TCP/IP usually fall into one of two major categories: The host could be *client* or the host could be a *server*. However, this is mostly an application-layer model issue because most computers are fully multitasking-capable today. It is possible that the same host could be running the client version of a program for one application (e.g., the Web browser) and the server version of another program (e.g., a file transfer server) at the same time. Dedicated servers are most common on the Internet, but almost all client computers can act as servers for a variety of applications.The details are not as important as the interplay among layers and applications.

Peer-to-Peer Models

The client–server model is not the only way to implement a protocol stack. Many applications implement a peer-to-peer model. Peer applications have exactly the same capabilities whether used as a client or as a server. Distributed file-sharing systems on the Internet typically function as both client (fetching files for the user) and as a server (allowing user files to be shared by others).

The differences between client–server and peer-to-peer models are mainly application layer differences. A desktop computer that runs a Web browser and has file sharing turned on is both client and server, but is not now peer-to-peer. As an aside, in X-windows, which is not discussed in this book, the terms "client" and "server" are actually reversed and users sit in front of "X-servers" and access "X-clients."

TCP/IP LAYERS AND CLIENT–SERVER

TCP/IP has five layers. The bottom layers are the physical layer and underlying network layer. The underlying network technologies at the network layer are the topic of the next chapter. Above the data link layer is the IP layer itself. The IP layer forms and routes the IP packet (also called a datagram in a lot of documentation) and IP is the major protocol at this layer.

The transport layer of TCP/IP consists of two major protocols: the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). TCP is a *reliable* layer added on top of the *best-effort* IP layer to make sure that even if packets are lost in transit, the hosts will be able to detect and resend missing information. TCP data units are called *segments*. UDP is as best-effort as IP itself, and UDP data units are called *datagrams*. The messages that applications exchange are made up of strings of segments or datagrams. Segments and datagrams are used to chop up application content, such as huge, multimegabyte files, into more easily handled pieces.

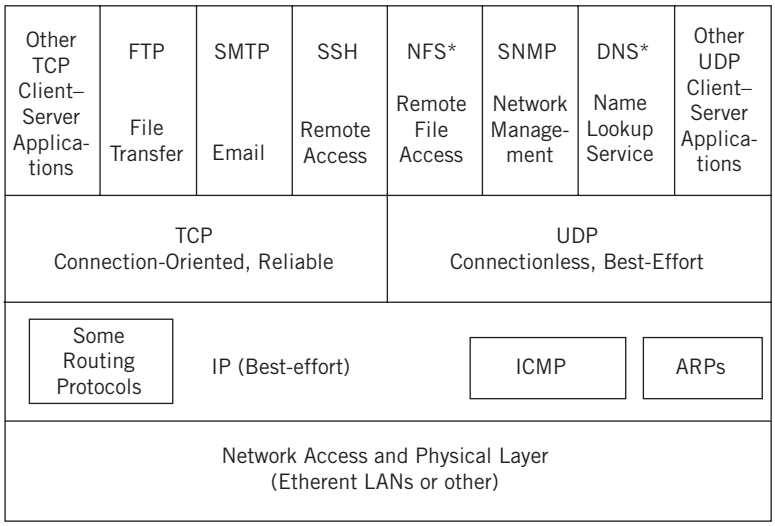
TCP is reliable in the sense that TCP always resends corrupt or lost segments. This strategy has many implications for delay-sensitive applications such as voice or video.

TCP is a *connection-oriented layer* on top of the *connectionless* IP layer. This means that before any TCP segment can be sent to another host, a TCP connection must be established to that host. Connectionless IP has no concept of a connection, and simply forwards packets without any understanding if the packets ever really got where they were going.

In contrast to TCP, UDP is a connectionless transport layer on top of connectionless IP. UDP segments are simply forwarded to a destination under the assumption that sooner or later a response will come back from the remote host. The response forms an implied or formal acknowledgment that the UDP segment arrived.

At the top of the TCP/IP stack is the application, or application services, layer. This is where the client–server concept comes into play. The applications themselves typically come in client or server versions, which is not true at other layers of TCP/IP. While a host computer might be able to run client processes and server processes at the same time, in the simplest case, these processes are two different applications.

Client–server application implementation can be extremely simple. A server process can start and basically sit and "listen" for clients to "talk" to the server. For example, a Web server is brought up on a host successfully whether there is a browser client pointed at it or not. The Web server process issues a *passive open* to TCP/IP and essentially remains idle on the network side until some client requests content. However, the Web browser (the client) process issues an *active open* to TCP/IP and attempts to



*In some instances, NFS and DNS use TCP.

FIGURE 2.4

The TCP/IP protocol stack in detail. The many possible applications on top and many possible network links on the bottom all funnel through the IP "hourglass."

send packets to a Web site immediately. If the Web site is not reachable, that causes an error condition.

To sum up the simplest application cases: Clients talk and servers listen (and usually reply). It is very easy to program an application that either talks or listens, although TCP/IP specifications allow for the transition of passive and active open from one state to another. We'll talk more about client and server application and passive and active opens in the chapter on sockets.

A more detailed look at the TCP/IP protocol stack is shown in Figure 2.4. The TCP/IP stack bridges the gap between interface connector on the network side (hardware) and the memory address space of the application on the host (software).

The names of the protocol data units used at each layer are worth reviewing. The unit of the network layer is the frame. Inside the frame is the data unit of the IP layer, the packet. The unit of the transport layer is the segment in TCP and datagram in UDP. The segment or datagram by definition is the content of the information-bearing packet. Finally, applications exchange messages. Segments and datagrams taken together form the messages that the applications are sending to each other.

This is a good place to explore some of the operational aspects of the TCP/IP protocol stack above the network access (or data link) layer.

THE IP LAYER

The connectionless IP layer routes the IP packets independently through the collection of network nodes such as routers that make up the "internetwork" that connects the LANs. Packets at the IP layer do not follow "paths" or "virtual circuits" or anything else set up by signaled or manually defined connections for packet flow in other types of network layers. However, this also means that the packets' content might arrive out of sequence, or even with gaps in the sequence due to lost packets, at the destination.

IP does not care to which application a packet belongs. IP delivers all packets without a sense of priority or sensitivity to loss. The whole point of IP is to get packets from one network interface to another. IP itself is not concerned with the lack of guaranteed *quality of service* (QoS) parameters such as bandwidth availability or minimal delay, and this is characteristic of all connectionless, best-effort networks. Even the basics, such as sequenced delivery of packet content, priorities, and guaranteed delivery in the form of acknowledgments (if these are needed by the application), must be provided by the higher layers of the TCP/IP protocol stack. These reliable transport functions are not functions of the IP layer, and some are not even functions of TCP.

Two other major protocols run at the IP layer besides IPv4 or IPv6 (or both). The routers that form the network nodes in a TCP/IP network must be able to send error messages to the hosts if a router must discard a packet (e.g., due to lack of buffer space because of congestion). This protocol is known as the Internet Control Message Protocol (ICMP). ICMP messages are sent inside IP packets, but ICMP is still considered a different protocol and not a separate layer.

The other major protocol placed at the IP layer has many different functions depending on the type of network that IP is running on. This is the Address Resolution Protocol (ARP). The main function of ARP is to provide a method for IPv4, which technically knows only about packets, to find out the proper network layer address to place in the frame header destination field. On LANs, this is the MAC address. Without this address, the network beneath the IP layer could not deliver the frame containing the IP packet to the proper destination. (IPv6 does not use ARP: IPv6 uses multicast for this purpose.)

On a LAN, ARP is a way for IPv4 to send a broadcast message onto the LAN asking, in effect, "Who has IP address 192.168.13.84?" Each system, whether host or router, on the LAN will examine the ARP message (all systems must pay attention to a broadcast) and the system having the IP address in question will reply to the sender's MAC address found in the source field of the frame. This target system will also cache the IP address information so that it knows the MAC address of the sender (this cuts down on ARP traffic on the network). The MAC layer address needed by the sending system is found in the source address field of the frame carrying the ARP reply packet.

ARP messages are broadcast to every host in what is called the network layer *broadcast domain*. The broadcast domain can be a single physical group (e.g., all hosts attached to a single group of hubs) or a logical grouping of hosts forming a *virtual LAN* (VLAN). More will be said about broadcast domains and VLANs later in this chapter.

THE TRANSPORT LAYER

The two main protocols that run above the IP layer at the transport layer are TCP and UDP. Lately, UDP has been assuming more and more prominence on the Internet, especially with applications such as voice and multicast traffic such as video. One reason is that TCP, with its reliable resending, is not particularly well suited for *real-time* applications (real time just means that the network delays must be low and stable or else the application will not function properly). For these applications, late-arriving data are worse than data that do not arrive at all, especially if the late data cause all the data "behind" it to also arrive late. (Of course, in spite of these limitations, TCP is still widely used for audio streaming and similar applications.)

Transmission Control Protocol

TCP's built-in reliability features include sequence numbering with resending, which is used to detect and resend missing or out-of-sequence segments. TCP also includes a complete flow control mechanism (called *windowing*) to prevent any sender from overwhelming a receiver. Neither of these built-in TCP features is good for real-time audio and video on the Internet. These applications cannot "pause" and wait for missing segments, nor should they slow down or speed up as traffic loads vary on the Internet. (The fact that they do just points out the incomplete nature of TCP/IP when it comes to quality of service for these applications and services.)

TCP contains all the functions and mechanisms needed to make up for the best-effort connectionless delivery provided by the IP layer. Packets could arrive at a host with errors, out of their correct sequence, duplicated, or with gaps in sequence due to lost (or discarded) packets. TCP must guarantee that the data stream is delivered to the destination application error-free, with all data in sequence and complete. Following the practice used in connection-oriented networks,TCP uses acknowledgments that periodically flow from the destination to the source to assure the sender that all is well with the data received to that point in time.

On the sending side, TCP passes segments to the IP layer for encapsulation in packets, which the IP layer in hosts and routers route connectionlessly to the destination host. On the receiving side,TCP accepts the incoming segments from the IP layer and delivers the data they represent to the proper application running above TCP in the exact order in which the data were sent.

User Datagram Protocol

The TCP/IP transport layer has another major protocol. UDP is as connectionless as IP. When applications use UDP instead of TCP, there is no need to establish, maintain, or tear down a connection between a source and destination before sending data. Connection management adds overhead and some initial delay to the network. UDP is a way to send data quickly and simply. However, UDP offers none of the reliability services that TCP does. UDP applications cannot rely on TCP to ensure error-free, guaranteed (via acknowledgments), in-sequence delivery of data to the destination.

For some simple applications, purely connectionless data delivery is good enough. Single request–response message pairs between applications are sent more efficiently with UDP because there is no need to exchange a flurry of initial TCP segments to establish a connection. Many applications will not be satisfied with this mode of operation, however, because it puts the burden of reliability on the application itself.

UDP is often used for short transactions that fit into one datagram and packet. Real-time applications often use UDP with another header inside called the real-time transport protocol (RTP). RTP borrows what it needs from the TCP header, such as a sequence number to detect (but not to resend) missing packets of audio and video, and uses these desirable features in UDP.

THE APPLICATION LAYER

At the top of the TCP/IP protocol stack, at the application layer, are the basic applications and services of the TCP/IP architecture. Several basic applications are typically bundled with the TCP/IP software distributed from various sources and, fortunately, are generally interoperable.

The standard application services suite usually includes a file transfer method (File Transfer Protocol: FTP), a remote terminal access method (Telnet, which is not commonly used today, and others, which are), an electronic mail system (Simple Mail

Transfer Protocol: SMTP), and a Domain Name System (DNS) resolver for domain name to IP address translation (and vice versa), and more. Many TCP/IP implementations also include a way of accessing files remotely (rather than transferring the whole file to the other host) known as the Network File System (NFS). There is also the Simple Network Management Protocol (SNMP) for network operations. For the Web, the server and browser applications are based on the Hypertext Transfer Protocol (HTTP). Some of these applications are defined to run on TCP and others are defined to run on UDP, and in many cases can run on either.

BRIDGES, ROUTERS, AND SWITCHES

The TCP/IP protocol stack establishes an architecture for internetworking. These protocols can be used to connect LANs in the same building, on a campus, or around the world. Not all internetworking devices are the same. Generally, network architects seeking to extend the reach of a LAN can choose from one of four major interconnection devices: repeaters, bridges, routers, and switches.

Not long ago, the network configuration and the available devices determined which type of internetworking device should be used. Today, network configurations are growing more and more complex, and the devices available often combine the features of several of these devices. For example, the routers on the Illustrated Network have all the features of traditional routers, plus some switching capabilities.

In their simplest forms, repeaters, bridges, and routers operate at different layers of the TCP/IP protocol stack, as shown in Figure 2.5. Roughly, repeaters forward bits from one LAN segment to another, bridges forward frames, and routers forward packets.

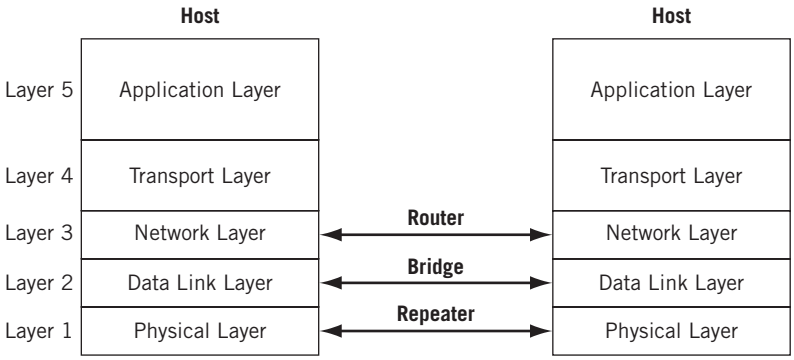


FIGURE 2.5

Repeater, bridge, and router. A repeater "spits bits," while a bridge deals with complete frames. A router operates at the packet level and is the main mode of the Internet.