Chapter 1

Communication Basics

- Analog and Digital Communications
- Communication Synchronization
- Physical and Logical Topologies
- The OSI Model
- Connection-Oriented and Connectionless Communications

BEFORE WE CAN DISCUSS routers and how they work, we first need to cover the basics. In this chapter, we will look at the fundamentals of network communications and how data is moved between systems. While the communication process is cloaked from the typical end user, a savvy network engineer must be armed with this information in order to be an effective troubleshooter.

We will start by looking at analog and digital signaling. All network communications rely on one of these transmission methods for moving information. We will then look at the kinds of problems that can occur during attempts to transmit information and how you can minimize the effects of these problems.

From there, we will talk about the core infrastructure of a network. We'll look at how systems get connected and exactly how digital or analog signaling is used to move information between systems. Finally, we'll map out the entire process of a communication session using the OSI model as a guide, so you can better understand exactly what is occurring on your network.

Analog and Digital Transmissions

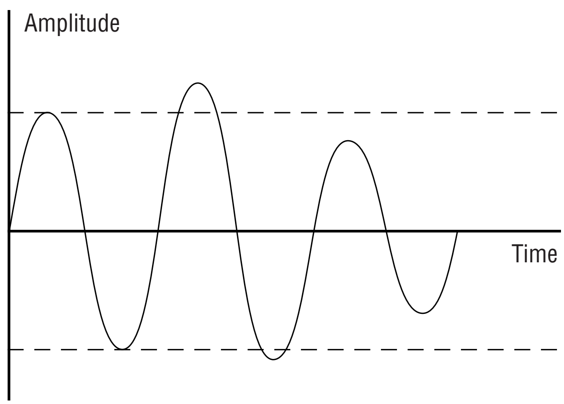
There are two ways data can be communicated:

- Through analog transmissions
- Through digital transmissions

An *analog transmission* is a signal that can vary either in power level (known as *amplitude*) or in the number of times this power level changes in a fixed period (known as *frequency*). An analog transmission can have a nearly infinite number of permissible values over a given range. For example, we use analog signals in order to communicate verbally. Our voice boxes vibrate the air at different frequencies and amplitudes. These vibrations are received by the eardrum and interpreted as words. Subtle changes in tone or volume can dramatically change the meaning of what we say.

Figure 1.1 shows an example of an analog transmission. Notice the amplitude each time the waveform peaks. Each of the three amplitude levels could be used to convey different information, such as alphanumeric characters. This makes for a very efficient way to communicate information, as each wave cycle can be used to convey additional information. In a perfect world, analog might be the ideal way to convey information.

FIGURE 1.1
An example of an analog transmission plotted over time



NOTE *Frequency is measured in cycles per second, or hertz (Hz). If Figure 1.1 were measured over a period of one second, it would be identified as a frequency of three cycles per second or 3Hz.*

The problem with analog transmissions is that they are very susceptible to *noise,* or interference. Noise is the addition of unwanted signal information. It can result in a number of data retransmissions, slowing down the rate of information transfer. Think of having a conversation in a crowded room with lots of people talking. With all of this background noise going on, it can become difficult to distinguish between your discussion and the others taking place within the room. Data retransmissions are signaled by phrases such as “What?” and “What did you say?” This slows down the rate of information transfer.

Figure 1.2 shows an example of an analog signal in a noisy circuit. Note that it is now more difficult to determine the precise amplitude of each waveform. This can result in incorrect information being transmitted or in requiring the correct information to be resent.

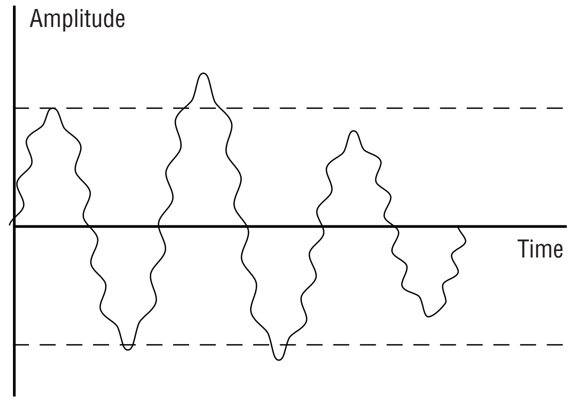


FIGURE 1.2
An analog transmission on a noisy circuit

To the rescue come *digital transmissions*. Digital communications are based on the binary system: Only two pieces of information are ever transmitted, a 1 or a 0. In an electrical circuit, a 0 is usually represented by a voltage of *zero* volts and a 1 is represented by five volts. This is radically different from analog transmissions, which can have an infinite number of possible values. These 1s and 0s are then strung together in certain patterns to convey information. For example, the binary equivalent of the letter *A* is `01000001`.

Each individual signal or digital pulse is referred to as a *bit*. When eight bits are strung together (like our binary equivalent of A), it is referred to as a *byte*. The byte is considered to be the base unit when dealing with digital communications. Each byte relays one complete piece of information, such as the letter A.

NOTE *Digital communication is analogous to Morse code or the early telegraph system: Certain patterns of pulses are used to represent different letters of the alphabet.*

If you examine Figure 1.3, you'll note that our waveform has changed shape. It is no longer a free-flowing series of arcs but now follows a rigid and predictable format.

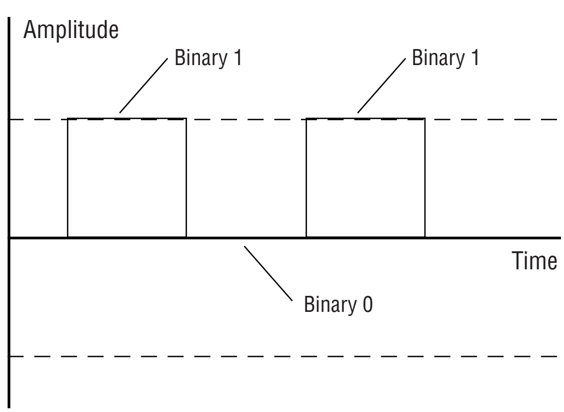
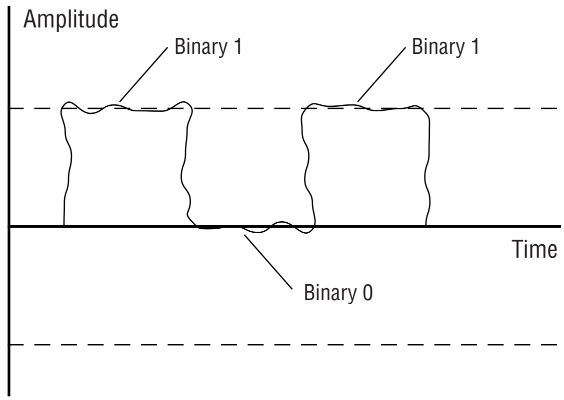


FIGURE 1.3
A digital transmission plotted over time

Because this waveform is so predictable and the variation between acceptable values is so great, it is now much easier to determine which values are being transmitted. As shown in Figure 1.4, even when there is noise in the circuit, you can still see which part of the signal is a binary 1 and which part is a 0.

FIGURE 1.4
A digital transmission on a noisy circuit



This simple format, which allows digital communication to be so noise resistant, can also be its biggest drawback. The information for the ASCII character A can be transmitted with a single analog wave or vibration, but transmitting the binary or digital equivalent requires eight separate waves or vibrations (to transmit `01000001`). Despite this inherent drawback, it is usually much more efficient to use digital communications whenever possible. Analog circuits require more overhead in order to detect and correct noisy transmissions. This is why most modern networks use digital communications.

NOTE Overhead *is the amount of additional information that must be transmitted on a circuit to insure that the receiving system gets the correct data and that the data is free of errors. Typically, when a circuit requires more overhead, less bandwidth is available to transmit the actual data. This is like the packaging used when something is shipped to you in a box. You didn't want hundreds of little Styrofoam peanuts, but they're there in the box taking up space to insure your item is delivered safely.*

Another big plus for digital communications is that computers process information in digital format. If you use analog communications to transfer information from one computer to another, you need some form of converter (such as a modem or a codex) at each end of the circuit to translate the information from digital to analog and then back to digital again.

Sources of Noise

So where does noise come from? Noise can be broken down into two categories:

- Electromagnetic interference (EMI)
- Radio frequency interference (RFI)

Electromagnetic Interference (EMI)

EMI is produced by circuits that use an alternating signal like analog or digital communications (referred to as an *alternating current* or an *AC circuit*). EMI is not produced by circuits that contain a consistent power level (referred to as a *direct current* or a *DC circuit*).

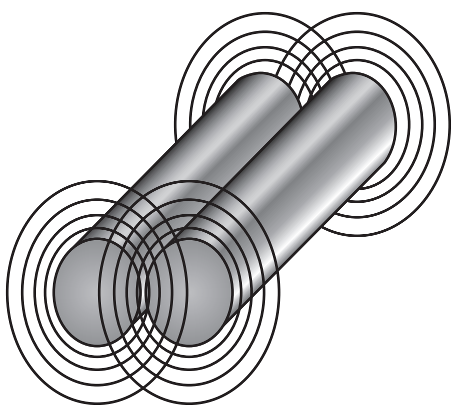
For example, if you could slice one of the wires coming from a car battery and watch the electrons moving down the wire (kids: don't try this at home), you would see a steady stream of power moving evenly and uniformly down the cable. The power level would never change; it would stay at a constant 12 volts. A car battery is an example of a DC circuit because the power level remains stable.

Now, let's say you could slice the wire to a household lamp and try the same experiment (kids: *definitely* do not try this at home!). You would now see that, depending on the point in time when you measured the voltage on the wire, it would read anywhere between –120 volts and +120 volts. The voltage level of the circuit is constantly changing. Plotted over time, the voltage level would resemble the analog signal shown earlier in Figure 1.1.

If you were to watch the flow of electrons now in the AC wire, you would notice something very interesting. As the voltage changes and the current flows down the wire, the electrons tend to ride predominantly on the surface of the wire. The center point of the wire would show almost no electron movement at all. If you increased the frequency of the power cycle, more and more of the electrons would travel on the surface of the wire instead of at the core. This effect is somewhat similar to what happens to a water skier—the faster the boat travels, the closer to the top of the water the skier rides.

As the frequency of the power cycle increases, energy begins to radiate at a 90° angle to the flow of current. Just as a water skier will push out wakes or waves, so too will energy move out from the center core of the wire. This radiation is in a direct relationship with the signal on the wire: If the voltage level or the frequency is increased, the amount of energy radiated will also increase (see Figure 1.5).

Figure 1.5
A conductor carrying an AC signal radiating EMI



This energy has magnetic properties and is the basis of how electromagnets and transformers operate. The downside to all of this is that the electromagnetic radiation can introduce an electrical signal into another wire if one is nearby. This interference either adds to or subtracts from the existing signal and is considered to be noise. EMI is the most common type of interference encountered on local area networks and can be produced by everything from fluorescent lights to network cables to heavy machinery. EMI also causes signal loss. Any energy that is dissipated as EMI is energy that can no longer be used to carry the signal down the wire.

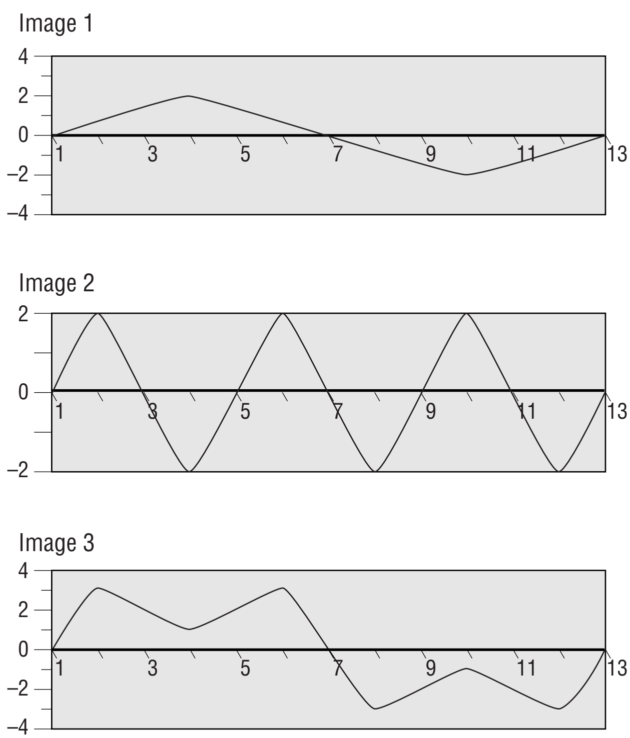
Radio Frequency Interference (RFI)

Radio frequency interference (RFI) can be produced when two signals have similar properties. The waveforms can join together, changing the frequency or amplitude of the resulting signal. This is why geographically close radio stations do not transmit on adjacent frequencies. If they did, a radio might not be able to receive the weaker of the two stations.

For an example, examine Image 1 in Figure 1.6. Assume that this is a communication signal we are transmitting between two systems. Now, let's assume that Image 2 is RFI that has been introduced to the circuit. These two signals would combine to produce the transmission shown in Image 3. Note that this is so far off from our original signal that our data would probably be incomprehensible.

Figure 1.6

The effects of RFI



The most common source of RFI in networking is caused by a condition known as *reflection*. Reflection occurs when a signal is reflected back upon itself by some component along its connection path. For example, a faulty connector within a circuit may reflect back some of the signal's energy to the original transmitting host. This is why all end points in a network must be capable not only of receiving the signal, but also of absorbing all of the signal's energy.

Communication Synchronization

Another important property in communications is letting the receiving system know when to expect data transmissions. If a receiving system cannot determine the beginning of a transmission, that system may mistake the beginning of a transmission for the middle or vice versa. This is true for both analog and digital communications.

Time Division

One way to achieve proper signal timing is to have the systems synchronize their communications so that each transmits data at a predetermined time. For example, the two systems may agree to take turns transmitting for one second each and then pass control over to the other system (similar to the give-and-take of a human conversation). This type of communication is known as *time division*, because the window of time when transmission is allowed is divided between the two systems.

While this type of negotiation is simple and straightforward, it has a number of inherent flaws. First, if a station has nothing to say, its time slice will be wasted while the second station sits by idly, waiting to transmit additional information. Also, if the stations' clocks are slightly different, the two systems will eventually fall out of sync and will smother each other's communication. Finally, consider what happens when further stations are plugged into the same circuit and have something to say: The time slices *could* be renegotiated, but this will severely diminish the amount of data that can be transmitted on this circuit in a timely fashion.

Despite its weaknesses, time division communication is used quite effectively by many wide area network (WAN) technologies. This is because a WAN circuit is typically between only two hosts. This eliminates the problem of trying to scale time division to many systems. Also, the fact that time division allocates bandwidth in such a predictable manner allows it to be an effective means of transmitting time-sensitive data such as video or voice.

The Preamble

To resolve the scaling problems with time division, many networking technologies communicate using a *preamble*: a defined series of communication pulses that tell all receiving stations, "Get ready—I've got something to say."

Using a preamble allows systems on the network to take a more ad hoc approach to communications. Instead of having to wait for their time slots to arrive, systems are allowed to attempt transmission anytime data must be conveyed. The preamble insures that all stations are able to sync up and receive the data in the same time measure that it was sent. This is just like a band's lead singer or drummer calling out the beat to lead into the start of a song, making sure all band members start the first note at exactly the same time and are in sync with each other.

Because a station sends a preamble only when it needs to transmit data, this eliminates dead-air time by leaving the circuit open for systems that need it. Also, keeping the data transmission bursts fairly small resolves the issue of systems falling out of sync due to time variations, because the stations can resync their times during each data delivery.

Understanding Topologies

The *topology* of a network is the set of rules for physically connecting and communicating on a given network medium. When you decide on a particular topology for connecting your network systems, you will need to follow a number of specifications that tell you how the systems need to be wired together, what type of connectors to use, and even how these systems must speak to each other on the wire.

Topology is broken down into two categories:

- Physical
- Logical

Physical Topology

Physical topology refers to how the transmission media are wired together. There are four types of physical topology:

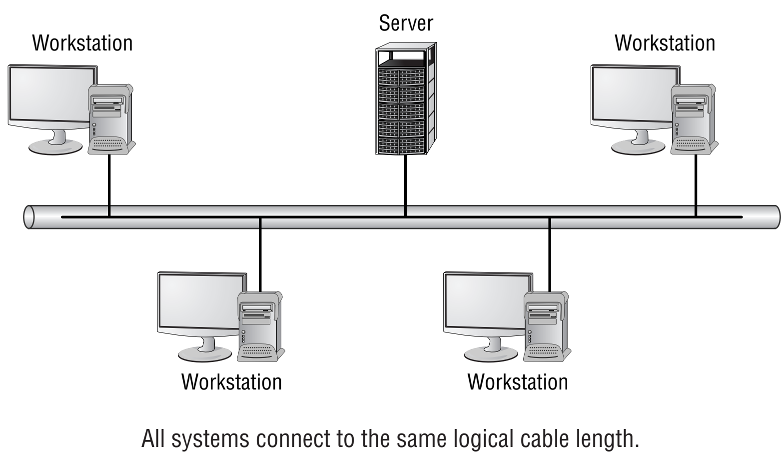
- Bus
- Star
- Ring
- Point to point

BUS TOPOLOGY

The *bus* topology is the common configuration for Thinnet wiring. Systems attached to the bus are connected in a series type of connection. All systems are connected via a single long cable run and tap in via T connectors. Figure 1.7 shows an example of a bus topology.

FIGURE 1.7

An example of a bus topology

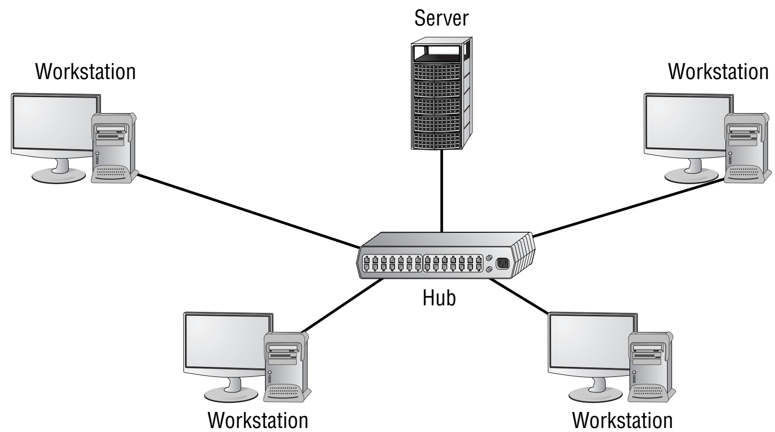


Star Topology

The *star* topology is the common configuration of twisted-pair wiring. Each system is connected to a central device, such as a hub or a switch. Only one system is connected to each physical wire run. These hubs and switches can then be linked together to form larger networks. Figure 1.8 shows an example of a star topology.

Figure 1.8

An example of a star topology

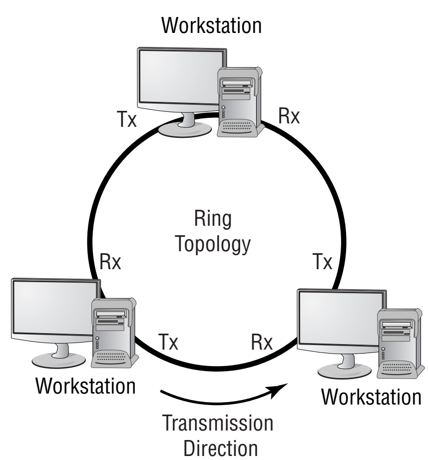


Ring Topology

The *ring* configuration is commonly used in token-based communications such as FDDI. The output data port (Tx for transmit) is connected to the input data port (Rx for receive) of the next station along the ring. This continues until the last station connects its output data port to the input data port of the first system, forming a complete ring. Figure 1.9 is an example of a ring topology.

Figure 1.9

An example of a ring topology



POINT TO POINT

A *point-to-point* connection (Figure 10.10) is commonly used in WAN configurations or in home networks with only two computers. With point to point, only two systems are connected to the physical medium. Fiber cable is commonly deployed in a point-to-point fashion. Twisted pair can also be configured for point-to-point connections by using a *crossover cable*. A crossover cable is simply a twisted-pair cable that has the transmit and receive pairs switched at one end.

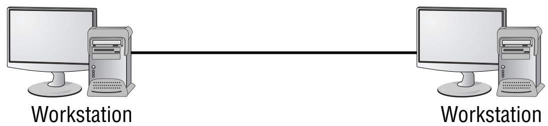


FIGURE 1.10
A point-to-point connection

NOTE *The transmission medium is separate from the physical topology. The examples I've just given are what you will commonly run into in the field, but they are not hard-and-fast rules. For example, even though fiber is commonly used in a ring topology, you can use it in a star or even a bus topology.*

PHYSICAL TOPOLOGIES AND CISCO ROUTERS

So what role does the physical topology play in deploying your Cisco routers? You need to determine up front what kind of physical topology you will be using in order to insure that you order a model which supports the right type of connectors.

For example, let's say you decide to use fiber optic cables to connect your Cisco router in order to support long cable runs. Cisco routers support two types of fiber optic connectors: SMA and FDDI. An SMA *connector* is commonly used in point-to-point applications. The FDDI *connector*, however, is commonly used in ring topologies. You need to determine which physical topology you will be using before selecting a Cisco model.

Logical Topology

A *logical topology* describes the communication rules each station should use when communicating on a network. For example, the specifications of the logical topology describe how each station should determine whether it's OK to transmit data, and what a station should do if it tries to transmit data at the same time as another station. The logical topology's job is to insure that information gets transferred as quickly and with as few errors as possible. Think of a discussion group moderator and you'll get the idea. The moderator insures that each person in the group gets a turn to speak. The moderator also insures that if two individuals try to speak at the same time, one gets priority and the other waits his or her turn.

So how are physical and logical topologies related? Any given logical topology will operate only on specific physical topologies. For example, Ethernet will operate on a bus, star, or point-to-point physical topology, but it will not work on a ring. The FDDI specification will function on a ring or a star topology but not on a bus or a point to point. Once you have determined which logical topology you will use, you can then go about selecting your physical topology.

Logical topologies are defined by the Institute of Electrical and Electronics Engineers (IEEE). The IEEE is a not-for-profit organization that consists of an assembly of companies and private individuals within the networking industry. The members of the IEEE work together to define specifications, preventing any single company from claiming ownership of the technology and helping to insure that products from multiple vendors will interoperate successfully in a network.

Table 1.1 shows the most common network specifications.

TABLE 1.1: COMMON IEEE NETWORK SPECIFICATIONS

Specification	Defines
IEEE 802.1	VLANs and bridging
IEEE 802.2	Logical link control (LLC)
IEEE 802.3	10Mb Ethernet
IEEE 802.3u	100Mb Ethernet
IEEE 802.3x	Flow Control
IEEE 802.3z	1Gb Ethernet (fiber)
IEEE 802.3ab	1Gb Ethernet (twisted pair)
IEEE 802.3ae	10 Gb Ethernet
IEEE 802.5	Token Ring
IEEE 802.7	Broadband
IEEE 802.11	Wireless Local Area Networks
IEEE 802.12	Demand priority
IEEE 802.14	Cable modem
IEEE 802.15	Wireless Personal Area Networks
IEEE 802.16	Broadband wireless

As a major player in the internetworking arena, Cisco has taken an active role in finalizing many of the specifications shown in Table 1.1. This not only helps to insure that Cisco products adhere to the IEEE specifications; it also helps to insure that support can be included as soon as a specification is ready for general consumption.

Connection Types

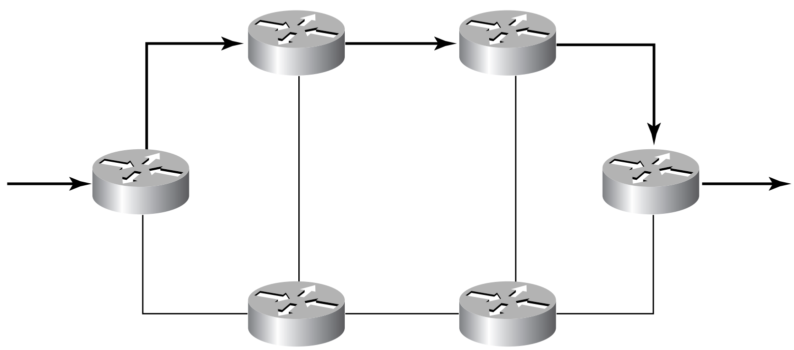
Every logical topology uses one of three methods for creating the connections between end stations:

- Circuit switching
- Message switching
- Packet switching

Circuit Switching

Circuit switching means that when data needs to be transferred from one node to another, a dedicated connection is created between the two systems. Bandwidth is dedicated to this communication session and remains available until the connection is no longer required. A regular telephone call uses circuit switching. When you place a call, a connection is set up between your phone and the one you are calling. This connection remains in effect until you finish your call and hang up. Figure 1.11 illustrates a circuit-switched network. The best route is selected, and bandwidth is dedicated to this communication session the entire length of the circuit, remaining in place until no longer needed. All data follows the same path.

FIGURE 1.11
An example of a circuit-switched network



Circuit-switched networks are useful for delivering information that must be received in the order it was sent. For example, applications such as real-time audio and video cannot tolerate the delays incurred in reassembling the data in the correct order. While circuit switching insures that data is delivered as quickly as possible by dedicating a connection to the task, it can also be wasteful compared to other types of connections, because the circuit will remain active even if the end stations are not currently transmitting.

Examples of circuit-switched networks include the following:

- Asynchronous Transfer Mode (ATM)
- Analog dial-up line (public telephone network)
- ISDN
- Leased line
- T1

Message Switching

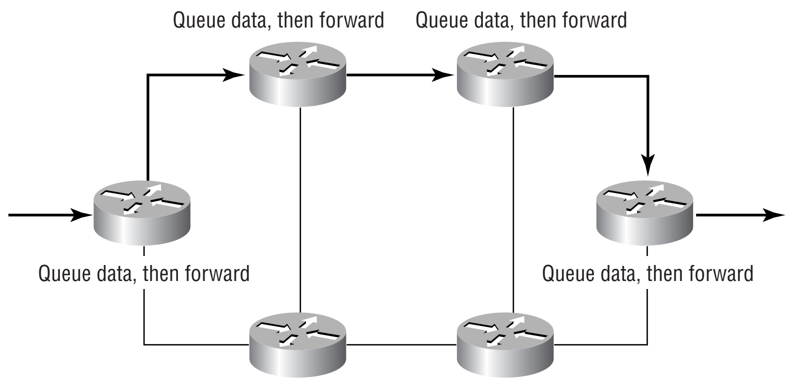
Message switching means that a *store-and-forward* type of connection is set up between connectivity devices along the message path. The first device creates a connection to the next and transmits the entire message. Once this transmission is complete, the connection is torn down, and the second device repeats the process if required.

The delivery of e-mail is a good example of message switching. As you type in your e-mail message, your computer queues the information until you are done. When you hit the Send button, your system delivers your message in its entirety to your local post office, which again queues the message. Your post office then contacts the post office of the person to whom you have addressed the message. Again, the message is delivered in its entirety and queued by the receiving system. Finally, the remote post office delivers your message to its intended recipient using the same process.

Figure 1.12 illustrates a message-switched network. While all the data still follows the same path, only one portion of the network is dedicated to delivering this data at any given time.

FIGURE 1.12

An example of a message-switched network



None of the logical topologies covered in this book uses message switching for the delivery of data. In part, this is because message switching increases the memory and processing requirements on interim hardware in order to store the information prior to delivery.

Packet Switching

The final method for connecting end stations is *packet switching*. This method is by far the most widely used in current networking topologies. Within a packet-switching network, each individual frame can follow a different path to its final destination. Because each frame can follow a different path, frames may or may not be received in the same order they were transmitted. To correct this problem, the receiving station uses the sequence numbers on the frames to reassemble the data in the correct order.

Note the operative phrase "can follow a different path." Other factors, such as the routing protocol, play a part in determining whether this feature is exploited. For now, however, it is enough to realize that in a packet-switched network all the data may not follow the same path.

Figure 1.13 illustrates a packet-switched network. Data is allowed to follow any path to its destination. Packet switching does not require that any bandwidth be reserved for this transmission.

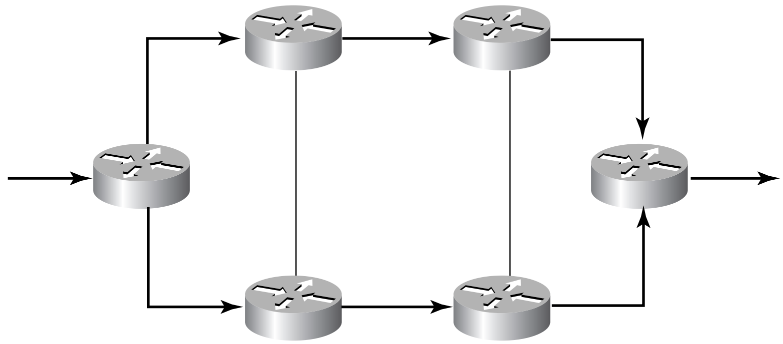


Figure 1.13
An example of a packet-switched network

Packet-switched networks are useful for transmitting regular network data. This includes storing files, printing, or cruising the Web. In short, all the activities you would normally associate with network usage will run fine in a packet-switched network. While packet switching is a poor choice for the delivery of live audio and video, it is extremely efficient for delivering information that is not time sensitive, because it does not require dedicating bandwidth to the delivery of information. Other nodes are capable of sharing the available bandwidth as required.

Here are some examples of packet-switched networks:

- All Ethernet topologies
- FDDI
- Frame Relay and X.25

Data Packaging

So far, we have talked about analog and digital signaling. We have also talked about physical and logical topologies and how they are used to tie our network together. It is now time to combine signaling with topologies in an attempt to transmit information between two systems.

When data is moved along a network, it is packaged inside a delivery envelope known as a *frame*. Frames are topology specific. An Ethernet frame needs to convey different information than a Token Ring or an ATM frame. Since Ethernet is by far the most popular topology, we will cover it in detail here.

Ethernet Frames

An *Ethernet frame* is a set of digital pulses transmitted onto the transmission media in order to convey information. An Ethernet frame can be anywhere from 64 to 1,518 bytes in size (a byte being eight digital pulses or bits) and is organized into four sections:

- Preamble
- Header
- Date
- Frame check sequence (FCS)

Preamble

Discussed earlier in this chapter, the preamble is used to synchronize communications between multiple systems along the same logical network. In an Ethernet environment, systems may begin transmitting at any time. The preamble allows systems receiving the transmission to get ready for the actual flow of data. An Ethernet preamble is eight bytes long.

NOTE *Because the preamble is considered part of the communication process and not part of the actual information being transferred, it is not usually included when measuring a frame's size.*

Header

A *header* always contains information about who sent the frame and where it is going. It may also contain other information, such as how many bytes the frame contains; this is called the *length field* and is used for error correction. If the receiving station measures the frame to be a different size than that indicated in the length field, it asks the transmitting system to send a new frame. If the length field is not used, the header may instead contain a *type field* that describes what type of Ethernet frame it is.

NOTE *The header size is always 14 bytes.*

Data

The *data* section of the frame contains the actual data the station needs to transmit, as well as any protocol information, such as source and destination IP addresses. The data field can be anywhere from 46 to 1,500 bytes in size. If a station has more than 1,500 bytes of information to transfer, it will break up the information over multiple frames and identify the proper order by using *sequence numbers*. Sequence numbers identify the order in which the destination system should reassemble the data. This sequence information is also stored in the data portion of the frame.

If the frame does not have 46 bytes' worth of information to convey, the station pads the end of this section by filling it in with 1s (remember that digital connections use binary numbers). Depending on the frame type, this section may also contain additional information describing what protocol or method of communication the systems are using.

Frame Check Sequence (FCS)

The *frame check sequence* insures that the data received is actually the data sent. The transmitting system processes the FCS portion of the frame through an algorithm called a *cyclic redundancy check* (CRC). This CRC takes the values of the above fields and creates a four-byte number. When the destination system receives the frame, it runs the same CRC and compares it to the value within this field. If the destination system finds a match, it assumes the frame is free of errors and processes the information. If the comparison fails, the destination station assumes that something happened to the frame during its travels and requests the transmitting system to send another copy of the frame.

NOTE *The FCS size is always four bytes.*

The Frame Header Section

Now that we have a better understanding of what an Ethernet frame is, let's take a closer look at the header section. The header information is ultimately responsible for identifying who sent the data and where the sender wanted it to go.

The header contains two fields to identify the source and the destination of the transmission. These are the *node addresses* of both the source and destination systems. This number is also referred to as the *media access co*ntrol (MAC) address. The node address is a unique number that is used to serialize network devices (like network cards or networking hardware) and is a unique identifier that distinguishes a given network device from any other network device in the world. No two network devices should ever be assigned the same number. Think of this number as equivalent to a telephone number. Every home with a telephone has a unique telephone number, so that the telephone company knows where to direct the call. In this same fashion, a local system will use the destination system's MAC address to send the frame to the proper system.

Note *The MAC address has nothing to do with Apple's computers and is always capitalized. It is the number used by each system attached to the network (PCs and Macs included) to uniquely identify itself.*

The MAC address is a six-byte, 12-digit hexadecimal number that is broken up into two parts. The first half of the address is the manufacturer's identifier. A manufacturer is assigned a range of MAC addresses to use when serializing its devices. Some of the more prominent MAC addresses appear in Table 1.2.

Table 1.2: Common MAC Addresses

First Three Bytes of MAC Address	Manufacturer
00000C	Cisco
0000A2	Bay Networks
0080D3	Shiva
00AA00	Intel
02608C	3Com
080009	Hewlett-Packard
080020	Sun
08005A	IBM

Tip *The first three bytes of the MAC address can be a good troubleshooting aid. If you are investigating a problem, try to determine the source MAC address. Knowing who made the device may put you a little closer to determining which system is giving you trouble. For example, if the first three bytes are* `00000C`*, you know you need to focus your attention on any Cisco devices on your network.*

The second half of the MAC address is the serial number the manufacturer has assigned to the device.

One address worthy of note is `FF-FF-FF-FF-FF-FF`. This is referred to as a broadcast *address*. A broadcast address is special: It means that all systems receiving this packet should read the included data. If a system sees a frame that has been sent to the broadcast address, it will read the frame and process the data if it can.

Note *You should never encounter a frame that has a broadcast address in the source node field. The Ethernet specifications do not include any conditions where the broadcast address should be placed in the source node field.*

Note that we already have address information and the capability of transferring information on our Ethernet network, yet we've made no mention of protocols. The reasons for this will become clearer in the next section when we discuss the *Address Resolution Protocol* (ARP). For now, remember that every system on our Ethernet segment sees every packet and needs to look at that packet to see whether or not the packet is addressed to that system.

If I am using a PC that only speaks IPX to a NetWare server, and somewhere on my network are two Apple computers speaking AppleTalk, my system still sees those frames and needs to look at every one of them to determine whether it needs to read the data within the frame. The fact that my system speaks a different protocol makes no difference. The Ethernet communication rules require that every computer on the segment look at every packet.

Note *Ethernet communication rules are discussed in greater detail in Chapter 2.*

That a computer must dedicate some CPU time to analyzing frames on a network may seem a minor point, but it isn't: If the network is busy, a workstation can appear to respond sluggishly, even though it is not intentionally transmitting or receiving network data.

Here's one last point about Ethernet frames before we move on. We have seen that each frame contains a 14-byte header and a four-byte FCS. These field lengths are fixed and never change. The sum of the two is 18 bytes. The data field, however, is allowed to vary from 46 to 1,500 bytes. This is where our minimum and maximum frame sizes come from:

```
46 + 18 = 64 bytes (minimum frame size)
1,500 + 18 = 1,518 bytes (maximum frame size)
```

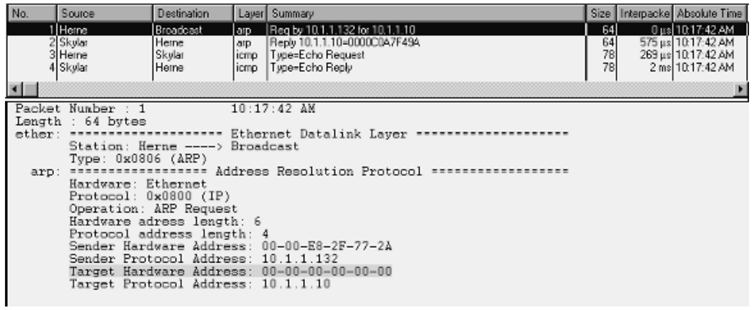
The Address Resolution Protocol

How do you find the destination MAC address so that you can send data to a system? After all, network cards do not ship with telephone books. Finding a MAC address is done with a special frame referred to as an *address resolution protocol* (ARP) frame. ARP functions differently depending on which protocol you're using (such as IPX, IP, NetBEUI, and so on).

For an example, see Figure 1.14. This is a decode of the initial packet from a system that wishes to send information to another system on the same network. Notice the information included within the decode. The transmitting system knows the IP address of the destination system, but it does not know the destination MAC address. Without this address, local delivery of data is not possible. ARP is used when a system needs to discover the destination system's MAC address.

Note *A frame* decode *is the process of converting a binary frame transmission to a format that can be understood by a human being. Typically, this is done using a network analyzer.*

Figure 1.14
A transmitting system attempting to discover the destination system's MAC address

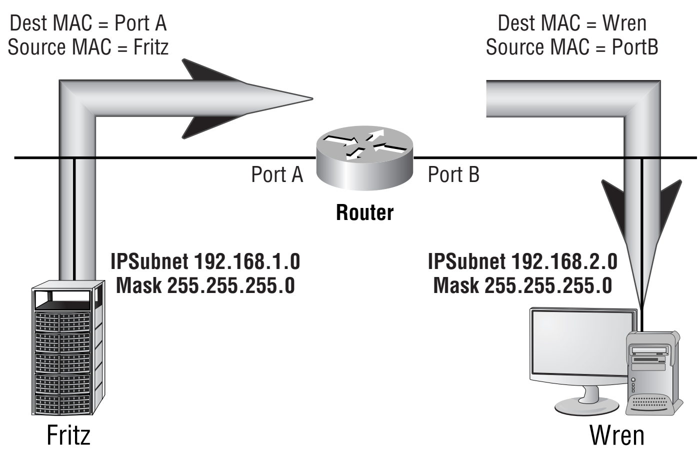


Keep in mind that ARP is only for local communications. When a packet of data crosses a router, the Ethernet header will be rewritten so that the source MAC address is that of the router, not the transmitting system. This means that a new ARP request may need to be generated.

ARP in Action

Figure 1.15 shows how this works. Our transmitting system (Fritz) needs to deliver some information to the destination system (Wren). Since Wren is not on the same subnet as Fritz, Fritz transmits an ARP in order to discover the MAC address of Port A on the local router. Once Fritz knows this address, Fritz transmits its data to the router.

Figure 1.15
MAC addresses are used for local communications only.



Our router will then need to send an ARP from Port B in order to discover the MAC address of Wren. Once Wren replies to this ARP request, the router will strip off the Ethernet frame from Fritz's data and create a new one. The router replaces the source MAC address (originally Fritz's MAC address) with the MAC address of Port B. It will also replace the destination MAC address (originally Port A) with the MAC address of Wren.

Note *When Fritz realized that Wren was not on the same subnet, he went looking for a router. We will discuss why in greater detail when we discuss networking protocols. For now, it is enough to understand that when two systems are in the same logical network, the MAC address is used to move data between systems.*

The ARP Cache

All systems are capable of caching information learned through ARP requests. For example, if a few seconds later Fritz wishes to send another packet of data to Wren, he would not have to transmit a new ARP request for the router's MAC address, as this value would be saved in memory. This memory area is referred to as the ARP cache.

ARP cache entries are retained for up to 60 seconds. After that, they are typically flushed out and must again be learned through a new ARP request. It is also possible to create static ARP entries, which creates a permanent entry in the ARP cache table. This way, a system is no longer required to transmit ARP requests for nodes with a static entry.

For example, we could create a static ARP entry for the router on Fritz's machine so that it would no longer have to transmit an ARP request when looking for this device. The only problem would occur if the router's MAC address changed. If the router were to fail and you had to replace it with a new one, you would also have to go back to Fritz's system and modify the static ARP entry, because the new router would have a different MAC address.

The OSI Model

In 1977, the International Organization of Standards (IOS) developed the *Open Systems Interconnection Reference Model* (OSI model) to help improve communications between different vendors' systems. The IOS was a committee representing many different organizations, whose goal was not to favor a specific method of communication but to develop a set of guidelines that would allow vendors to insure that their products would interoperate.

The IOS was setting out to simplify communications between systems. Many events must take place in order to insure that data first reaches the correct system and is then passed along to the correct application in a usable format. A set of rules was required to break down the communication process into a simple set of building blocks.

SIMPLIFYING A COMPLEX PROCESS

An analogy to the OSI model would be the process of building a house. While the final product may seem a complex piece of work, it is much simpler when it is broken down into manageable sections.

A good house starts with a foundation. There are rules that define how wide the foundation wall must be, as well as how far below the frost line it needs to sit. After that, the house is framed off. Again, there are rules to define how thick the lumber must be and how far each piece of framing can span without support. Once the house is framed, there is a defined process for putting on a roof, adding walls, and connecting the electrical system and plumbing.

By breaking down this complicated process into small, manageable sections, building a house becomes easier. This breakdown also makes it easier to define who is responsible for which section. For example, the electrical contractor's responsibilities include running wires and adding electrical outlets, but not shingling the roof.

The entire structure becomes an interwoven tapestry, with each piece relying on the others. For example, the frame of our house requires a solid foundation. Without it, the frame will eventually buckle and fall. The frame may also require that load-bearing walls be placed in certain areas of the house in order to insure that the frame does not fall in on itself.

The OSI model strives to set up similar kinds of definitions and dependencies. Each portion of the communication process becomes a separate building block. This makes it easier to determine what each portion of the communication process is required to do. It also helps to define how each piece will be connected to the others.

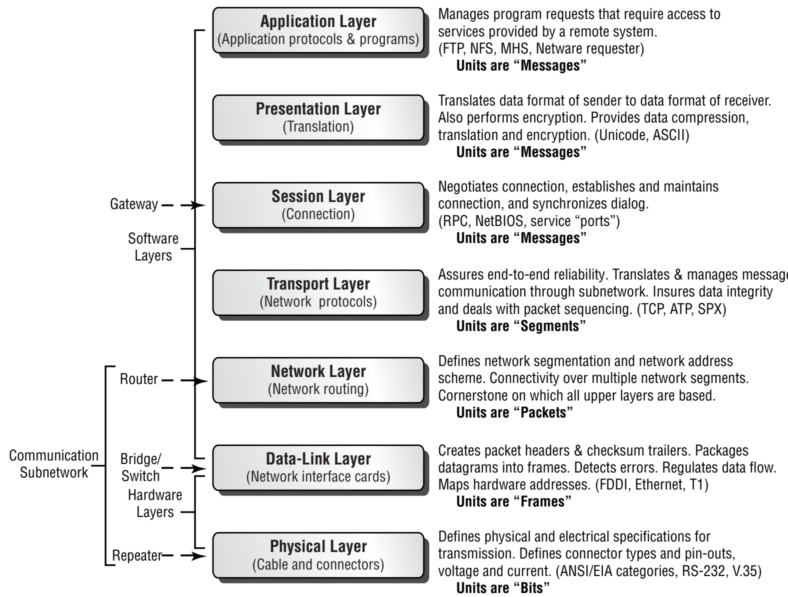
The OSI Layers Defined

The OSI model consists of a set of seven layers. Each layer describes how its portion of the communication process should function, as well as how it will interface with the layers directly above it, below it, and adjacent to it on other systems. This allows a vendor to create a product that operates on a certain level and to be sure it will operate in the widest range of applications. If the vendor's product follows a specific layer's guidelines, it should be able to communicate with products, created by other vendors, that operate at adjacent layers.

To return to our house analogy for just a moment, think of the lumberyard that supplies main support beams used in house construction. As long as the yard follows the guidelines for thickness and material, builders can expect beams to function correctly in any house that has a proper foundation structure.

Figure 1.16 represents the OSI model in all its glory. Let's take the layers one at a time to determine the functionality expected of each.

Figure 1.16
The OSI model



Layer 1: The Physical Layer

The *Physical layer* describes the specifications of our transmission media, connectors, and signal pulses. Choosing to use analog or digital signaling would be considered a Physical layer specification. So would the medium that carries these signals, such as twisted pair, fiber, or even the atmosphere.

Network hubs and repeaters are referred to as Physical layer devices. This is because they are little more than signal amplifiers. All of a hub's functionality is defined within the first layer of the OSI model.

Layer 2: The Data-Link Layer

The *Data-Link* layer describes the specifications for topologies and the communication between local systems. Ethernet is a good example of a Data-Link layer specification because it is capable of functioning with multiple Physical layer specifications (such as twisted pair and fiber cabling) as well as with multiple Network layer specifications (such as IP, IPX, and AppleTalk).

The Data-Link layer is the "door between worlds," connecting the physical aspects of the network (cables and digital pulses) with the abstract world of software and data streams. Bridges and switches are considered data-link devices because they are capable of controlling traffic based on topology address information. For example, in an Ethernet environment the source and destination MAC addresses can be used to control traffic flow.

Note *Topologies such as Ethernet are discussed in Chapter 2.*

Layer 3: The Network Layer

The *Network layer* describes how systems on different network segments find each other; it also defines network addresses. IP, IPX, and AppleTalk's Datagram Delivery Protocol (DDP) are all examples of Network layer specifications because they define a mechanism for finding distant resources as well as addressing individual systems.

Note *Protocols are discussed in Chapter 3.*

Layer 4: The Transport Layer

The *Transport layer* deals with the actual manipulation of your data and prepares it for delivery through the network. If your data is too large for a single frame, the Transport layer breaks it up into smaller pieces and assigns sequence numbers. Sequence numbers allow the Transport layer on the receiving system to reassemble the data into its original content. While the Data-Link layer performs a CRC check on all frames, the Transport layer can act as a backup check to insure that all the data was received and is usable.

Examples of Transport layer functionality would be IP's Transmission Control Protocol (TCP), User Datagram Protocol (UDP), IPX's Sequence Packet Exchange (SPX), and AppleTalk's AppleTalk Transaction Protocol (ATP). Note that these specifications also have components that would be considered part of the Session layer, as well.

Note *Transport layer functionality is discussed further in the "Transport Layer Services" section of this chapter.*

Layer 5: The Session Layer

The *Session layer* deals with establishing and maintaining a connection between two or more systems. It insures that a query for a specific type of service is made correctly. For example, if you try to access a system with your Web browser, the Session layers on both systems work together to insure you receive HTML pages and not e-mail. If a system is running multiple network applications, it is up to the Session layer to keep these communications orderly and to insure that incoming data is directed to the correct application.

Layer 6: The Presentation Layer

The *Presentation layer* insures that data is received in a format that is usable to applications running on the system. For example, if you are communicating over the Internet using encrypted communications, the Presentation layer would be responsible for encrypting and decrypting this information. Most Web browsers support this kind of functionality for performing financial transactions over the Internet. Data and language translations also occur at this level.

Layer 7: The Application Layer

The label *Application layer* is a bit misleading, because this term does not describe the actual program that a user may be running on his system. Rather, this is the layer that is responsible for determining when access to network resources is required. For example, Microsoft Word does not function at the Application layer of the OSI model. If a user tries to retrieve a document from their home directory on a server, however, the Application layer networking software is responsible for delivering that request to the remote system.

Note *In geek lingo, the layers are numbered in the order I've described them. If I were to state that switches function at layer 2 of the OSI model, you would interpret this to mean that switches work within the guidelines provided by the Data-Link layer of the OSI model.*

How the OSI Model Works

When data is transmitted between systems, it is the job of each OSI layer to communicate with

- The layer above it
- The layer below it
- The adjacent layer on the remote system

For example, the Network layer on a transmitting host should be able to communicate with its Data-Link and Transport layer counterparts. It should also be able to communicate with the Network layer on the remote system.

Let's look at an example to see how these layers work together. Assume you're using your word processor program, and you want to retrieve a file called `resume.txt` from your home directory on a remote server. The networking software running on your system would react similarly to the description that follows.

Formulating a File Request

The Application layer detects that you are requesting information from a remote file system. It formulates a request to that system that `resume.txt` should be read from disk. Once it has created this request, the Application layer passes the request to the Presentation layer for further processing.

The Presentation layer determines whether it needs to encrypt this request or perform any type of data translation. Once this has been determined and completed, the Presentation layer then adds any information it needs to pass along to the Presentation layer on the server and forwards the packet down to the Session layer.

The Session layer checks which application is requesting the information and verifies the service being requested from the server (file access). The Session layer adds information to the request to insure that the remote system knows how to handle this request. Then, it passes all this information along to the Transport layer.

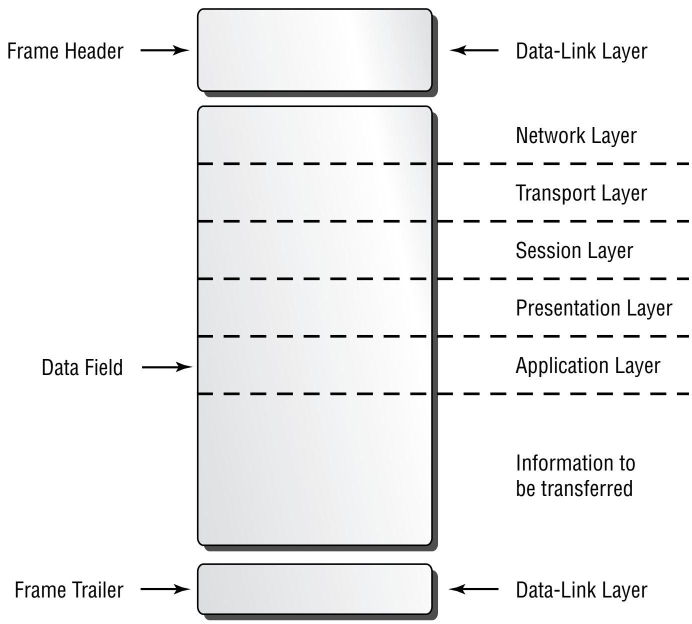
The Transport layer insures that it has a reliable connection to the server and begins the process of breaking down all the information so that it can be packaged into frames. If more than one frame is required, the information is split up and each block of information is assigned a sequence number. These sequenced chunks of information are passed one at a time to the Network layer.

The Network layer receives the blocks of information from the Transport layer and adds the network address for both the local workstation and the server. This happens to each block before it is passed down to the Data-Link layer.

At the Data-Link layer, the blocks are packaged into individual frames. Note that all the information added by each of the previous layers (as well as the actual file request) must fit into the 46- to 1,500-byte data field of the Ethernet frame. This is shown in Figure 1.17. The Data-Link layer then adds a frame header, which consists of the source and destination MAC addresses, and uses this information (along with the contents of the data field) to create a CRC trailer. The Data-Link layer is then responsible for transmitting the frame according to the topology rules in use on the network. Depending on the topology, this could mean listening for a quiet moment on the network, waiting for a token, or waiting for a specific time division before transmitting the frame.

FIGURE 1.17

The location of each layer's information within our frame



NOTE *The Physical layer does not add any information to the frame.*

The Physical layer is responsible for carrying the information from the source system to its destination. Because the Physical layer has no knowledge of frames, it is simply passing along the digital signal pulses transmitted by the Data-Link layer. The Physical layer is the medium by which a connection is made between the two systems; it is responsible for carrying the signal to the Data-Link layer on the remote system.

Our workstation has successfully formulated our data request ("Send me a copy of `resume.txt`") and transmitted it to the server. At this point, the server follows a similar process, but in reverse.

Receiving Data on the Server

The Data-Link layer on the server receives the transmitted frame. It notes that the MAC address in the destination field of the header is its own and recognizes that it needs to process this request. It performs a CRC check on the frame and compares the results to the value stored in the frame trailer. If these values match, the Data-Link layer strips off the header and trailer and passes the data field up to the Network layer. If the values do not match, the Data-Link layer sends a request to the source system asking that another frame be sent.

The Network layer on the server will analyze the information recorded by the Network layer on the workstation. It will note that the destination software address is its own. Once this analysis is complete, the Network layer removes information related to this level and passes the remainder up to the Transport layer.

The Transport layer receives the information and analyzes the information recorded by the Transport layer on the workstation. If it finds that packet sequencing was used, it will queue any information it receives until all the data has been received. If any of the data is missing, the Transport layer will use the sequence information to formulate a reply to the workstation, requesting that this piece of data be sent again. Once all the data has been received, the Transport layer will strip out any transport information and pass the full request up to the Session layer.

The Session layer will receive the information and verify that it is from a valid connection. If the check is positive, the Session layer strips out any session information and passes the request up to Presentation layer.

The Presentation layer receives the frame and analyzes the information recorded by the Presentation layer on the workstation. It then performs any translation or decryption required. Once translation or decryption has been completed, it strips out the Presentation layer information and passes the request up to the Application layer.

The Application layer insures that the correct process running on the server receives the request for data. Because this is a file request, it is passed to whichever process is responsible for access to the file system.

This process then reads the requested file and passes the information back to the Application layer. At this point, the entire process of passing the information through each of the layers would repeat. If you're amazed that the requested file is retrievable in anything less than a standard coffee break, then you have a pretty good idea of the magnitude of what happens when you request a simple file.

Cisco Routers and the OSI Model

Since a router controls traffic at the Network layer, it is considered an OSI layer 3 device. A Cisco router does offer, however, some higher-level services. For example, a Cisco router can control traffic flow using information contained in the Transport and Session layers. You can also use Telnet to remotely access the router. Telnet would be considered a function of layer 7 on the OSI model.

NOTE *While a router is predominately a layer 3 device, do not forget about the additional functionality that has been included. This will become especially important when it comes time to focus on security.*

Transport Layer Services

Now we can get our information from point A to point B, regardless of whether the systems are located on the same logical network. This raises the question, "Once we get there, how do we carry on a proper conversation?" This is where the Transport layer comes in.

The Transport layer is where we begin to set down the rules of communication etiquette. It's not enough that we can get this information from one system to another; we also have to insure that both systems are operating at the same level of decorum.

As an analogy, let's say you pull up to the finest restaurant in the city in your GMC Pacer and proceed to the front door donning your best set of leather chaps, Harley jacket, and bandanna. Once inside, you greet the maitre d' by stating, "Yo, wimp, gimme a table and some grub, NOW!" Surprisingly, you're escorted out of the restaurant at gunpoint. What went wrong? Why, you used improper etiquette, of course. Everyone knows the correct term is not "grub" but "escargot."

You can avoid such verbal breakdown, as well as those in network communications, by insuring that all parties involved are conversing at the same level of etiquette. There are two forms of network communication etiquette:

- Connection-oriented
- Connectionless

Connection-Oriented Communication

A *connection-oriented communication* exchanges control information, called a *handshake*, before transmitting data. The Transport layer uses the handshake to insure that the destination system is ready to receive information. A connection-oriented exchange will also insure that data is transmitted and received in its original order.

Modems are heavy users of connection-oriented communications, as they need to negotiate a connection speed before sending any information. In networking, this functionality is accomplished through the use of a Transport layer field, which is referred to as a *flag* in the IP and AT world, or as the *connection control field* under IPX. Only connection-oriented communications use these fields. When IP is the underlying routing protocol, TCP is used to create connection-oriented communications. IPX uses SPX, and AppleTalk uses ATP to provide this functionality. As a communication session is started, the Application layer (not necessarily the program you are using) will specify whether it needs to use a connection-oriented protocol. Telnet is just such an application. When a Telnet session is started, the Application layer will request TCP as its transport service to better insure reliability of the connection. Let's look at how this session is established to see how a handshake works.

The TCP Three-Packet Handshake

At your workstation, you type in **telnet thor.foobar.com** to establish a remote connection to that system. As the request is passed down through the Transport layer, TCP is selected to connect the two systems so that a connection-oriented communication can be established. The Transport layer sets the synchronization (SYN) flag to 1 and leaves all other flags at 0. IP uses multiple flag fields and uses the binary system to set values. This means that the only possible values of an IP flag are 1 and 0. IPX and AT use a hexadecimal value, as their frames only contain one flag field. This allows the one field to contain more than two values.

By setting SYN to 1 and all other fields to 0, you let the system on the other end (`thor.foobar.com`) know that you wish to establish a new communication session with the system. This request is then passed down the remaining OSI layers, across the wire to the remote system, and then up through its OSI layers.

If the Telnet service is available on the remote system, the request is acknowledged and sent back down the stack until it reaches the Transport layer. The Transport layer then sets the SYN flag to 1 as did the originating system, but it will also set the acknowledgment (ACK) flag to 1, as well. This lets the originating system know that its transmission was received and that it's OK to send data. The request is then passed down the stack and over the wire back to the original system.

The original system then sets the SYN flag to 0 and the ACK flag to 1 and transfers this frame back to Thor. This tells Thor, "I'm acknowledging your acknowledgment and I'm about to send data." At this point data is transferred, with each system required to acknowledge each packet it receives. Figure 1.18 shows a Telnet session from the system Loki to the system Thor. Each line represents a different frame that has been transmitted from one system to the other. Source and destination systems are identified, as well as some summary information about the frame. Notice that the first three frames are identified as TCP frames, not Telnet, and that they perform the handshaking described above. Once TCP establishes the connection-oriented connection, Telnet can step in to transfer the data required. The TCP frames that appear later in the conversation are for acknowledgment purposes. Remember that with a connection-oriented protocol, every frame must be acknowledged. If the frame was a request for information, the reply can be in the form of delivering the requested information.

Figure 1.18

An example of a connection-oriented communication

No.	Siz	Source	Destination	Layer	Summary
1	64	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	tcp	Port:1042 ---> TELNET SYN
2	64	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	tcp	Port:TELNET ---> 1042 ACK SYN
3	64	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	tcp	Port:1042 ---> TELNET ACK
4	82	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	telnt	Cmd=Do; Code=Suppress Go Ahead; Cmd=Will; Code=Termin
5	64	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	tcp	Port:TELNET ---> 1042 ACK
6	70	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	telnt	Cmd=Do; Code=Terminal Type; Cmd=Do; Code=Terminal Spe
7	64	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	telnt	Cmd=Won't; Code=; Cmd=Will; Code=Terminal Type;
8	73	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	telnt	Cmd=Will; Code=Suppress Go Ahead; Cmd=Do; Code=; Cmd=
9	64	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	tcp	Port:TELNET ---> 1042 ACK
10	67	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	telnt	Cmd=Subnegotiation Begin; Code=; Data=..P....
11	76	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	telnt	Cmd=Subnegotiation Begin; Code=Terminal Speed; Data= .....
12	64	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	tcp	Port:1042 ---> TELNET ACK
13	64	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	tcp	Port:TELNET ---> 1042 ACK
14	92	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	telnt	Cmd=Subnegotiation Begin; Code=Terminal Speed; Data= .38
15	64	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	telnt	Cmd=Do; Code=Echo;
16	64	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	telnt	Cmd=Won't; Code=Echo;
17	129	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	telnt	Cmd=Will; Code=Echo; Data=..Red Hat Linux release 4.1 (Var
18	64	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	telnt	Cmd=Do; Code=Echo;
19	64	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	tcp	Port:TELNET ---> 1042 ACK
20	65	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	telnt	Data=login:
21	64	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	tcp	Port:1042 ---> TELNET ACK

A Connection-Oriented Analogy

In case you're still a bit fuzzy on handshaking and connection-oriented communications, let's look at an analogy. Let's say you call a friend to inform him you'll be having a network Quake party on Saturday night and that he should come by with his laptop. You follow these steps:

- You dial your friend's phone number (SYN=1, ACK=0).
- Your friend answers the phone and says, "Hello" (SYN=1, ACK=1).
- You reply by saying, "Hi Fred, this is Dave" (SYN=0, ACK=1).

You would then proceed to transfer your data about your upcoming party. Every time you pause, your friend would either transfer back information ("Yes, I'm free Saturday night") or send some form of acknowledgment (ACK) to let you know he has not yet hung up.

When the conversation is complete, you would both tear down the connection by saying goodbye, which is a handshake to let each other know that the conversation is complete and that it's OK to hang up the phone.

The purpose of connection-oriented communications is simple. They provide a reliable communication session when the underlying layers may be considered less than stable. Insuring reliable connectivity at the Transport layer helps to speed up communication when data becomes lost. This is because the data does not have to be passed all the way up to the Application layer before a retransmission frame is created and sent. While this is important in modem communications, where a small amount of noise or a crossed line can kill a communication session, it is not as useful with network-based communication. TCP and SPX originate from the days when the Physical and Data-Link layers could not always be relied on to successfully transmit information. These days, this is less of a concern because reliability has increased dramatically from the earlier years of networking.

Connectionless Communication

A *connectionless* protocol requires neither an initial handshake nor that acknowledgments be sent for every packet. When you use a connectionless transport, it makes its best effort to deliver the data but relies on the stability of the underlying layers, as well as Application layer acknowledgments, to insure that the data is delivered reliably. IP's User Datagram Protocol (UDP) and IPX's NetWare Core Protocol (NCP) are examples of connectionless transports. Both protocols rely on connectionless communications to transfer routing and server information, as well. Although AppleTalk does not utilize any connectionless communications for creating data sessions, AppleTalk does use connectionless communications when advertising servers with its Name Binding Protocol (NBP).

Note *Broadcasts are always transmitted using a connectionless transport.*

A Connectionless Example

As an example of connectionless communications, check out the Network File System (NFS) session in Figure 1.19. NFS is a service that allows file sharing over IP. Its older versions used UDP as the underlying transport protocol. Note that all data acknowledgments are in the form of a request for additional information. The destination system (Thor) assumes that the last packet was received if the source system (Loki) requests additional information. Conversely, if Loki does not receive a reply from Thor for information it has requested, NFS takes care of requesting the information again. As long as you have a stable connection that does not require a large number of retransmissions, this is a very efficient method of communicating because it does not generate unnecessary acknowledgments.

FIGURE 1.19

NFS uses UDP to create a connectionless session.

No.	Size	Source	Destination	Layer	Summary
1	198	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Lookup ???/games.tar.gz
2	174	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Lookup for games.tar.gz
3	182	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Get File Attributes for games.tar.gz
4	142	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Get File Attributes
5	194	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Read From File games.tar.gz; Offset 0; 1024 bytes
6	1,170	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Read From File; 1024 bytes
7	194	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Read From File games.tar.gz; Offset 1024; 1024 bytes
8	1,170	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Read From File; 1024 bytes
9	194	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Read From File games.tar.gz; Offset 2048; 1024 bytes
10	1,170	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Read From File; 1024 bytes
11	194	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Read From File games.tar.gz; Offset 3072; 1024 bytes
12	1,170	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Read From File; 1024 bytes
13	194	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Read From File games.tar.gz; Offset 4096; 1024 bytes
14	1,170	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Read From File; 1024 bytes
15	194	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Read From File games.tar.gz; Offset 5120; 1024 bytes
16	1,170	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Read From File; 1024 bytes
17	194	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Read From File games.tar.gz; Offset 6144; 1024 bytes
18	1,170	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Read From File; 1024 bytes
19	194	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Read From File games.tar.gz; Offset 7168; 1024 bytes
20	1,170	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Read From File; 1024 bytes
21	194	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Read From File games.tar.gz; Offset 8192; 1024 bytes
22	1,170	THOR.FOOBAR.COM	LOKI.FOOBAR.COM	nfs	Reply Read From File; 1024 bytes
23	194	LOKI.FOOBAR.COM	THOR.FOOBAR.COM	nfs	Call Read From File games.tar.gz; Offset 9216; 1024 bytes

A Connectionless Analogy

Let's look at another analogy to see how this type of communication differs from the connection-oriented one described earlier. Again, let's say you call Fred to inform him you'll be having a network Quake party on Saturday night and that he should come by with his laptop. You call Fred's number, but this time you get his answering machine. You leave a detailed message indicating when the party will take place and what he should bring. Unlike the first call, which Fred himself answered, you are now relying on

- Your ability to dial the correct phone number, as you did not reach your friend to confirm that this number was in fact his
- The fact that the phone company did not drop your phone connection in the middle of your message (answering machines do not ACK—unless, of course, you talk until the beep cuts you off)
- The answering machine's proper recording of the message—without eating the tape
- The ability of Fred's cat to discern between the tape and a ball of yarn
- The absence of a power failure (which would cause the machine to lose the message)
- Fred's retrieval of this message between the time of your call and the date of the party

As you can see, you have no real confirmation that your friend will actually receive the message. You are counting on the power company, the answering machine, and so on, to enable Fred to receive your message in a timely manner. If you wanted to insure the reliability of this data transmission, you could send an Application layer acknowledgment request in the form of "Please RSVP by Thursday." If you did not get a response by then, you could try transmitting the data again.

The benefit of a connectionless protocol is that it allows for a bit more freedom when determining how nitpicky the systems must be to insure proper data delivery. As you will find in our discussion of IPX, this can be leveraged so that many frames of useful information can be sent before an acknowledgment of their receipt is required.

Connection-Oriented vs. Connectionless

So which is a better transport to use? Unfortunately, the answer is whichever one your Application layer specifies. If Telnet wants TCP, you cannot force it to use UDP. Even if you could, the receiving system would be expecting a TCP handshake.

When a network program is initially coded, it is up to the programmers to choose which transport the program will support. Most applications today use a connection-oriented transport. There are a couple of reasons for this.

The first reason involves conditions at the time of development. Telnet has been around for so long that TCP was the appropriate transport at the time of development. Networking was still a very unstable animal back then. Trying to switch over to UDP now is possible but would cause a complete mess as some systems are upgraded before others.

The other reason is a misunderstanding about when a connection-oriented protocol is required. A programmer who is faced with using a reliable or an unreliable transport for moving data will usually choose the former—without regard as to how inefficient this may make the communication session. Many programmers simply look at the term "unreliable" and shy away. If the programmer does not, certainly her pointy-haired boss will.

Summary

In this chapter, we covered the basics of network communication. We discussed the principles of transmission as well as how information gets transmitted from one system to another. We also discussed how the OSI model can help to simplify the somewhat complex process of network communication.

In the next chapter, we will take a closer look at logical topologies. We will look at the available options for both LAN and WAN applications, and how to choose the right topology for your specific needs.