CHAPTER 6

Bandwidth Utilization: Multiplexing and Spectrum Spreading

In real life, we have links with limited bandwidths. The wise use of these bandwidths has been, and will be, one of the main challenges of electronic communications. However, the meaning of *wise* may depend on the application. Sometimes we need to combine several low-bandwidth channels to make use of one channel with a larger bandwidth. Sometimes we need to expand the bandwidth of a channel to achieve goals such as privacy and antijamming. In this chapter, we explore these two broad categories of bandwidth utilization: multiplexing and spectrum spreading. In multiplexing, our goal is efficiency; we combine several channels into one. In spectrum spreading, our goals are privacy and antijamming; we expand the bandwidth of a channel to insert redundancy, which is necessary to achieve these goals.

This chapter is divided into two sections:

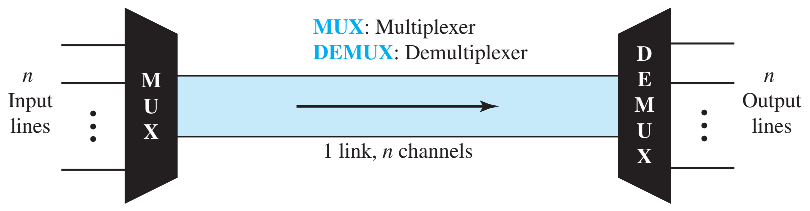
- The first section discusses multiplexing. The first method described in this section is called *frequency-division multiplexing* (FDM), which means to combine several analog signals into a single analog signal. The second method is called *wavelength-division multiplexing* (WDM), which means to combine several optical signals into one optical signal.The third method is called *time-division multiplexing* (TDM), which allows several digital signals to share a channel in time.
- The second section discusses spectrum spreading, in which we first spread the bandwidth of a signal to add redundancy for the purpose of more secure transmission before combining different channels. The first method described in this section is called *frequency hopping spread spectrum* (FHSS), in which different modulation frequencies are used in different periods of time. The second method is called *direct sequence spread spectrum* (DSSS), in which a single bit in the original signal is changed to a sequence before transmission.

6.1 MULTIPLEXING

Whenever the bandwidth of a medium linking two devices is greater than the bandwidth needs of the devices, the link can be shared. **Multiplexing** is the set of techniques that allow the simultaneous transmission of multiple signals across a single data link. As data and telecommunications use increases, so does traffic. We can accommodate this increase by continuing to add individual links each time a new channel is needed; or we can install higher-bandwidth links and use each to carry multiple signals. As described in Chapter 7, today's technology includes high-bandwidth media such as optical fiber and terrestrial and satellite microwaves. Each has a bandwidth far in excess of that needed for the average transmission signal. If the bandwidth of a link is greater than the bandwidth needs of the devices connected to it, the bandwidth is wasted. An efficient system maximizes the utilization of all resources; bandwidth is one of the most precious resources we have in data communications.

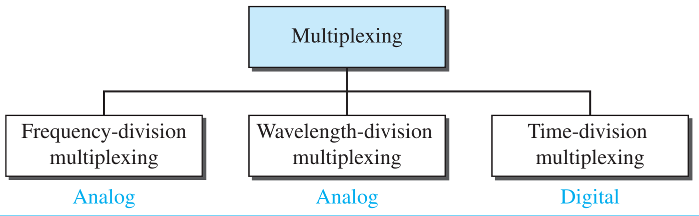
In a multiplexed system, *n* lines share the bandwidth of one link. Figure 6.1 shows the basic format of a multiplexed system. The lines on the left direct their transmission streams to a **multiplexer (MUX),** which combines them into a single stream (many-to-one). At the receiving end, that stream is fed into a **demultiplexer (DEMUX),** which separates the stream back into its component transmissions (one-to-many) and directs them to their corresponding lines. In the figure, the word **link** refers to the physical path. The word **channel** refers to the portion of a link that carries a transmission between a given pair of lines. One link can have many (*n*) channels.

Figure 6.1 *Dividing a link into channels*



There are three basic multiplexing techniques: frequency-division multiplexing, wavelength-division multiplexing, and time-division multiplexing. The first two are techniques designed for analog signals, the third, for digital signals (see Figure 6.2).

Figure 6.2 *Categories of multiplexing*


Multiplexing
Frequency-division multiplexing
Wavelength-division multiplexing
Time-division multiplexing
Analog
Analog
Digital

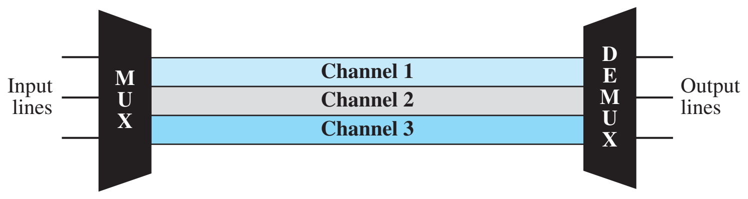
Although some textbooks consider carrier division multiple access (CDMA) as a fourth multiplexing category, we discuss CDMA as an access method (see Chapter 12).

6.1.1 Frequency-Division Multiplexing

Frequency-division multiplexing (FDM) is an analog technique that can be applied when the bandwidth of a link (in hertz) is greater than the combined bandwidths of the signals to be transmitted. In FDM, signals generated by each sending device modulate different carrier frequencies. These modulated signals are then combined into a single composite signal that can be transported by the link. Carrier frequencies are separated by sufficient bandwidth to accommodate the modulated signal. These bandwidth ranges are the channels through which the various signals travel. Channels can be separated by strips of unused bandwidth—**guard bands**—to prevent signals from overlapping. In addition, carrier frequencies must not interfere with the original data frequencies.

Figure 6.3 gives a conceptual view of FDM. In this illustration, the transmission path is divided into three parts, each representing a channel that carries one transmission.

Figure 6.3 *Frequency-division multiplexing*



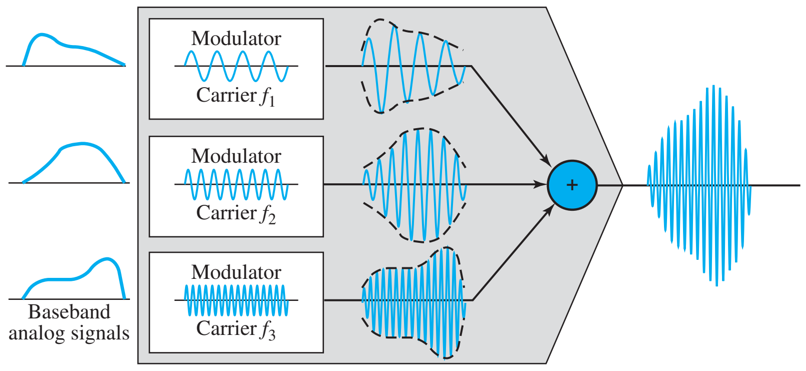
We consider FDM to be an analog multiplexing technique; however, this does not mean that FDM cannot be used to combine sources sending digital signals. A digital signal can be converted to an analog signal (with the techniques discussed in Chapter 5) before FDM is used to multiplex them.

FDM is an analog multiplexing technique that combines analog signals.

Multiplexing Process

Figure 6.4 is a conceptual illustration of the multiplexing process. Each source generates a signal of a similar frequency range. Inside the multiplexer, these similar signals modulate different carrier frequencies (f_1, f_2, and f_3). The resulting modulated signals are then combined into a single composite signal that is sent out over a media link that has enough bandwidth to accommodate it.

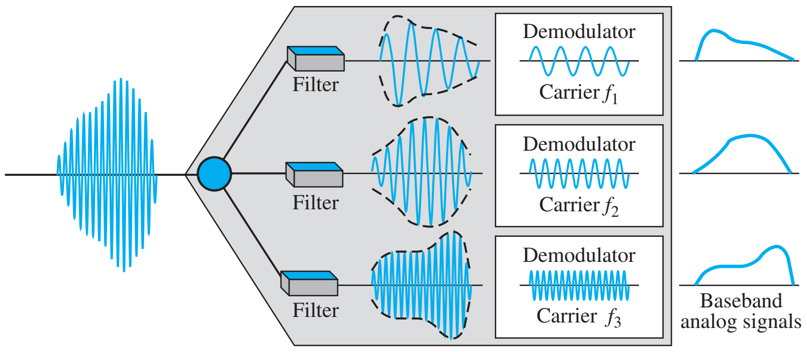
Figure 6.4 *FDM process*



Demultiplexing Process

The demultiplexer uses a series of filters to decompose the multiplexed signal into its constituent component signals. The individual signals are then passed to a demodulator that separates them from their carriers and passes them to the output lines. Figure 6.5 is a conceptual illustration of demultiplexing process.

Figure 6.5 *FDM demultiplexing example*



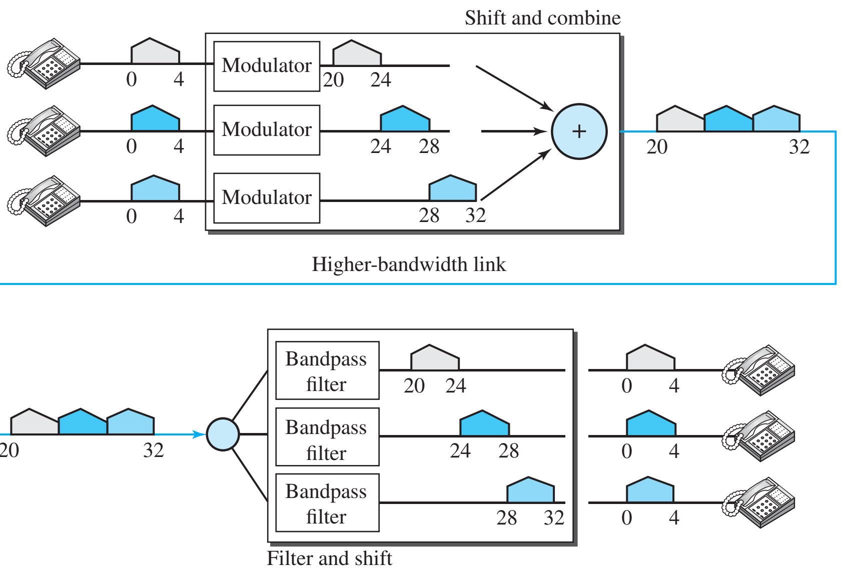
Example 6.1

Assume that a voice channel occupies a bandwidth of 4 kHz. We need to combine three voice channels into a link with a bandwidth of 12 kHz, from 20 to 32 kHz. Show the configuration, using the frequency domain. Assume there are no guard bands.

Solution

We shift (modulate) each of the three voice channels to a different bandwidth, as shown in Figure 6.6. We use the 20- to 24-kHz bandwidth for the first channel, the 24- to 28-kHz bandwidth

Figure 6.6 *Example 6.1*



for the second channel, and the 28- to 32-kHz bandwidth for the third one. Then we combine them as shown in Figure 6.6. At the receiver, each channel receives the entire signal, using a filter to separate out its own signal. The first channel uses a filter that passes frequencies between 20 and 24 kHz and filters out (discards) any other frequencies. The second channel uses a filter that passes frequencies between 24 and 28 kHz, and the third channel uses a filter that passes frequencies between 28 and 32 kHz. Each channel then shifts the frequency to start from zero.

Example 6.2

Five channels, each with a 100-kHz bandwidth, are to be multiplexed together. What is the minimum bandwidth of the link if there is a need for a guard band of 10 kHz between the channels to prevent interference?

Solution

For five channels, we need at least four guard bands. This means that the required bandwidth is at least $5 \times 100 + 4 \times 10 = 540$ kHz, as shown in Figure 6.7.

Example 6.3

Four data channels (digital), each transmitting at 1 Mbps, use a satellite channel of 1 MHz. Design an appropriate configuration, using FDM.

Solution

The satellite channel is analog. We divide it into four channels, each channel having a 250-kHz bandwidth. Each digital channel of 1 Mbps is modulated so that each 4 bits is modulated to 1 Hz. One solution is 16-QAM modulation. Figure 6.8 shows one possible configuration.

Figure 6.7 *Example 6.2*

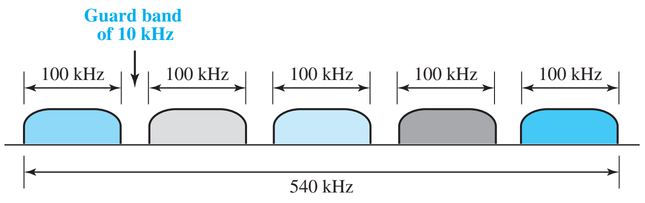
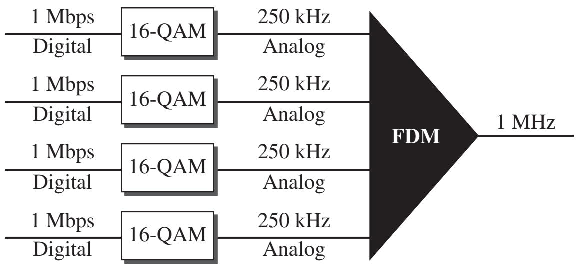


Figure 6.8 *Example 6.3*



The Analog Carrier System

To maximize the efficiency of their infrastructure, telephone companies have traditionally multiplexed signals from lower-bandwidth lines onto higher-bandwidth lines. In this way, many switched or leased lines can be combined into fewer but bigger channels. For analog lines, FDM is used.

One of these hierarchical systems used by telephone companies is made up of groups, supergroups, master groups, and jumbo groups (see Figure 6.9).

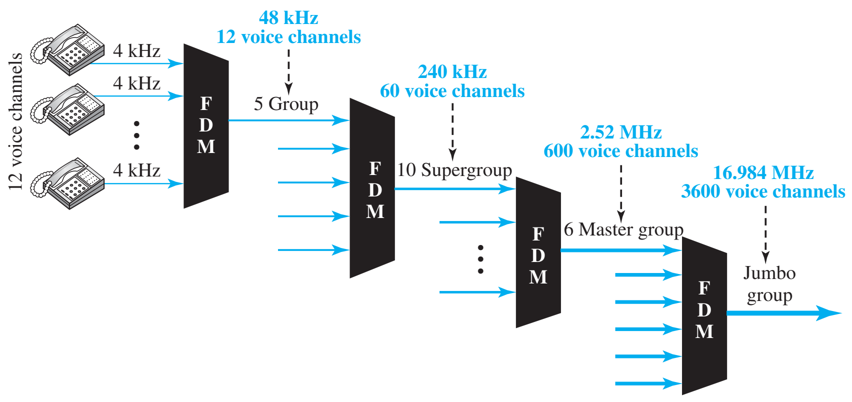
In this **analog hierarchy,** 12 voice channels are multiplexed onto a higher-bandwidth line to create a **group.** A group has 48 kHz of bandwidth and supports 12 voice channels.

At the next level, up to five groups can be multiplexed to create a composite signal called a **supergroup.** A supergroup has a bandwidth of 240 kHz and supports up to 60 voice channels. Supergroups can be made up of either five groups or 60 independent voice channels.

At the next level, 10 supergroups are multiplexed to create a **master group.** A master group must have 2.40 MHz of bandwidth, but the need for guard bands between the supergroups increases the necessary bandwidth to 2.52 MHz. Master groups support up to 600 voice channels.

Finally, six master groups can be combined into a **jumbo group.** A jumbo group must have 15.12 MHz (6×2.52 MHz) but is augmented to 16.984 MHz to allow for guard bands between the master groups.

Figure 6.9 *Analog hierarchy*



Other Applications of FDM

A very common application of FDM is AM and FM radio broadcasting. Radio uses the air as the transmission medium. A special band from 530 to 1700 kHz is assigned to AM radio. All radio stations need to share this band. As discussed in Chapter 5, each AM station needs 10 kHz of bandwidth. Each station uses a different carrier frequency, which means it is shifting its signal and multiplexing. The signal that goes to the air is a combination of signals. A receiver receives all these signals, but filters (by tuning) only the one which is desired. Without multiplexing, only one AM station could broadcast to the common link, the air. However, we need to know that there is no physical multiplexer or demultiplexer here. As we will see in Chapter 12, multiplexing is done at the data-link layer.

The situation is similar in FM broadcasting. However, FM has a wider band of 88 to 108 MHz because each station needs a bandwidth of 200 kHz.

Another common use of FDM is in television broadcasting. Each TV channel has its own bandwidth of 6 MHz.

The first generation of cellular telephones (See Chapter 16) also uses FDM. Each user is assigned two 30-kHz channels, one for sending voice and the other for receiving. The voice signal, which has a bandwidth of 3 kHz (from 300 to 3300 Hz), is modulated by using FM. Remember that an FM signal has a bandwidth 10 times that of the modulating signal, which means each channel has 30 kHz (10×3) of bandwidth. Therefore, each user is given, by the base station, a 60-kHz bandwidth in a range available at the time of the call.

Example 6.4

The Advanced Mobile Phone System (AMPS) uses two bands. The first band of 824 to 849 MHz is used for sending, and 869 to 894 MHz is used for receiving. Each user has a bandwidth of 30 kHz in each direction. The 3-kHz voice is modulated using FM, creating 30 kHz of modulated signal. How many people can use their cellular phones simultaneously?

Solution

Each band is 25 MHz. If we divide 25 MHz by 30 kHz, we get 833.33. In reality, the band is divided into 832 channels. Of these, 42 channels are used for control, which means only 790 channels are available for cellular phone users. We discuss AMPS in greater detail in Chapter 16.

Implementation

FDM can be implemented very easily. In many cases, such as radio and television broadcasting, there is no need for a physical multiplexer or demultiplexer. As long as the stations agree to send their broadcasts to the air using different carrier frequencies, multiplexing is achieved. In other cases, such as the cellular telephone system, a base station needs to assign a carrier frequency to the telephone user. There is not enough bandwidth in a cell to permanently assign a bandwidth range to every telephone user. When a user hangs up, her or his bandwidth is assigned to another caller.

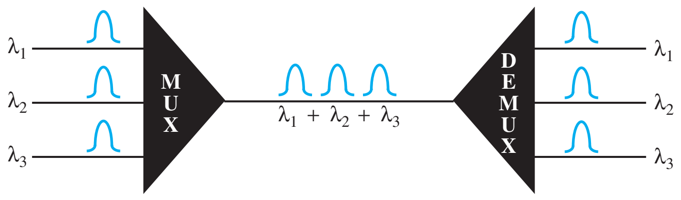
6.1.2 Wavelength-Division Multiplexing

Wavelength-division multiplexing (WDM) is designed to use the high-data-rate capability of fiber-optic cable. The optical fiber data rate is higher than the data rate of metallic transmission cable, but using a fiber-optic cable for a single line wastes the available bandwidth. Multiplexing allows us to combine several lines into one.

WDM is conceptually the same as FDM, except that the multiplexing and demultiplexing involve optical signals transmitted through fiber-optic channels. The idea is the same: We are combining different signals of different frequencies. The difference is that the frequencies are very high.

Figure 6.10 gives a conceptual view of a WDM multiplexer and demultiplexer. Very narrow bands of light from different sources are combined to make a wider band of light. At the receiver, the signals are separated by the demultiplexer.

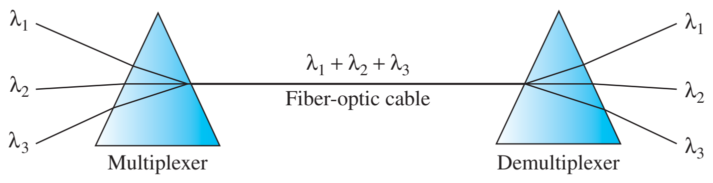
Figure 6.10 *Wavelength-division multiplexing*



WDM is an analog multiplexing technique to combine optical signals.

Although WDM technology is very complex, the basic idea is very simple. We want to combine multiple light sources into one single light at the multiplexer and do the reverse at the demultiplexer. The combining and splitting of light sources are easily handled by a prism. Recall from basic physics that a prism bends a beam of light based on the angle of incidence and the frequency. Using this technique, a multiplexer can be made to combine several input beams of light, each containing a narrow band of frequencies, into one output beam of a wider band of frequencies. A demultiplexer can also be made to reverse the process. Figure 6.11 shows the concept.

Figure 6.11 *Prisms in wavelength-division multiplexing and demultiplexing*



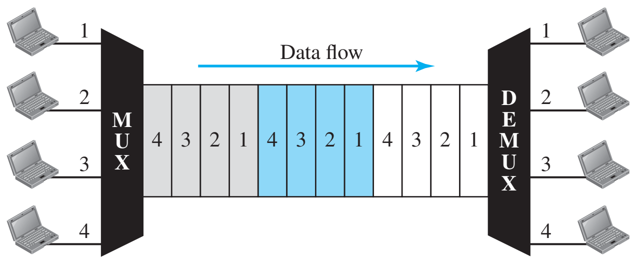
One application of WDM is the SONET network, in which multiple optical fiber lines are multiplexed and demultiplexed. We discuss SONET in Chapter 14.

A new method, called **dense WDM (DWDM),** can multiplex a very large number of channels by spacing channels very close to one another. It achieves even greater efficiency.

6.1.3 Time-Division Multiplexing

Time-division multiplexing (TDM) is a digital process that allows several connections to share the high bandwidth of a link. Instead of sharing a portion of the bandwidth as in FDM, time is shared. Each connection occupies a portion of time in the link. Figure 6.12 gives a conceptual view of TDM. Note that the same link is used as in FDM; here, however, the link is shown sectioned by time rather than by frequency. In the figure, portions of signals 1, 2, 3, and 4 occupy the link sequentially.

Figure 6.12 *TDM*



Note that in Figure 6.12 we are concerned with only multiplexing, not switching. This means that all the data in a message from source 1 always go to one specific destination, be it 1, 2, 3, or 4. The delivery is fixed and unvarying, unlike switching.

We also need to remember that TDM is, in principle, a digital multiplexing technique. Digital data from different sources are combined into one timeshared link. However, this does not mean that the sources cannot produce analog data; analog data can be sampled, changed to digital data, and then multiplexed by using TDM.

TDM is a digital multiplexing technique for combining several low-rate channels into one high-rate one.

We can divide TDM into two different schemes: synchronous and statistical. We first discuss **synchronous TDM** and then show how **statistical TDM** differs.

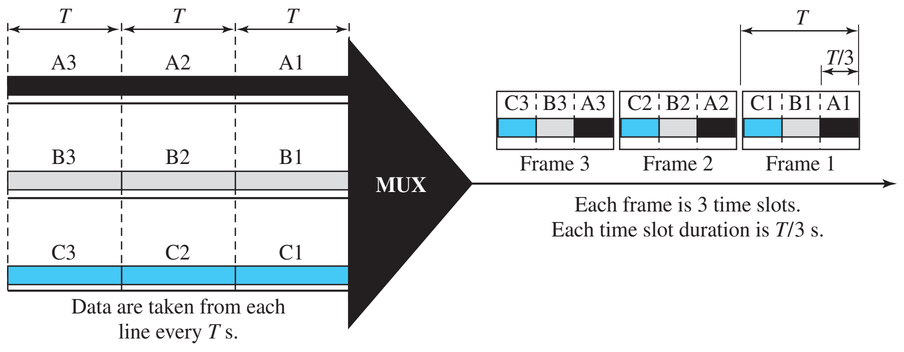
Synchronous TDM

In synchronous TDM, each input connection has an allotment in the output even if it is not sending data.

Time Slots and Frames

In synchronous TDM, the data flow of each input connection is divided into units, where each input occupies one input time slot. A unit can be 1 bit, one character, or one block of data. Each input unit becomes one output unit and occupies one output time slot. However, the duration of an output time slot is n times shorter than the duration of an input time slot. If an input time slot is T s, the output time slot is T/n s, where n is the number of connections. In other words, a unit in the output connection has a shorter duration; it travels faster. Figure 6.13 shows an example of synchronous TDM where n is 3.

Figure 6.13 *Synchronous time-division multiplexing*


T
T
T
A3
A2
A1
B3
B2
B1
C3
C2
C1
Data are taken from each line every T s.
MUX
T
T/3
C3 B3 A3
C2 B2 A2
C1 B1 A1
Frame 3
Frame 2
Frame 1
Each frame is 3 time slots.
Each time slot duration is T/3 s.

In synchronous TDM, a round of data units from each input connection is collected into a frame (we will see the reason for this shortly). If we have n connections, a frame is divided into n time slots and one slot is allocated for each unit, one for each input line. If the duration of the input unit is T, the duration of each slot is T/n and the duration of each frame is T (unless a frame carries some other information, as we will see shortly).

The data rate of the output link must be n times the data rate of a connection to guarantee the flow of data. In Figure 6.13, the data rate of the link is 3 times the data rate of a connection; likewise, the duration of a unit on a connection is 3 times that of the time slot (duration of a unit on the link). In the figure we represent the data prior to multiplexing as 3 times the size of the data after multiplexing. This is just to convey the idea that each unit is 3 times longer in duration before multiplexing than after.

In synchronous TDM, the data rate of the link is *n* times faster, and the unit duration is *n* times shorter.

Time slots are grouped into frames. A frame consists of one complete cycle of time slots, with one slot dedicated to each sending device. In a system with *n* input lines, each frame has *n* slots, with each slot allocated to carrying data from a specific input line.

Example 6.5

In Figure 6.13, the data rate for each input connection is 1 kbps. If 1 bit at a time is multiplexed (a unit is 1 bit), what is the duration of

1. each input slot,
2. each output slot, and
3. each frame?

Solution

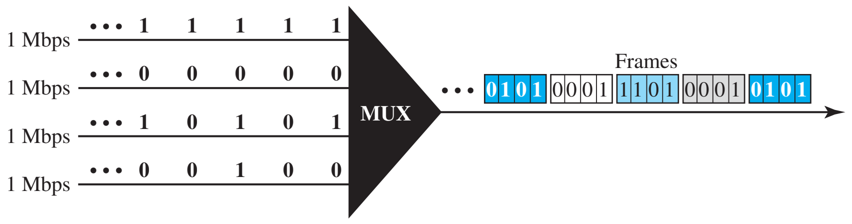
We can answer the questions as follows:

1. The data rate of each input connection is 1 kbps. This means that the bit duration is 1/1000 s or 1 ms. The duration of the input time slot is 1 ms (same as bit duration).
2. The duration of each output time slot is one-third of the input time slot. This means that the duration of the output time slot is 1/3 ms.
3. Each frame carries three output time slots. So the duration of a frame is 3 × 1/3 ms, or 1 ms. The duration of a frame is the same as the duration of an input unit.

Example 6.6

Figure 6.14 shows synchronous TDM with a data stream for each input and one data stream for the output. The unit of data is 1 bit. Find (1) the input bit duration, (2) the output bit duration, (3) the output bit rate, and (4) the output frame rate.

Figure 6.14 *Example 6.6*



Solution

We can answer the questions as follows:

1. The input bit duration is the inverse of the bit rate: 1/1 Mbps = 1 μs.
2. The output bit duration is one-fourth of the input bit duration, or 1/4 μs.

3. The output bit rate is the inverse of the output bit duration, or 1/4 µs, or 4 Mbps. This can also be deduced from the fact that the output rate is 4 times as fast as any input rate; so the output rate = 4 × 1 Mbps = 4 Mbps.
4. The frame rate is always the same as any input rate. So the frame rate is 1,000,000 frames per second. Because we are sending 4 bits in each frame, we can verify the result of the previous question by multiplying the frame rate by the number of bits per frame.

Example 6.7

Four 1-kbps connections are multiplexed together. A unit is 1 bit. Find (1) the duration of 1 bit before multiplexing, (2) the transmission rate of the link, (3) the duration of a time slot, and (4) the duration of a frame.

Solution

We can answer the questions as follows:

1. The duration of 1 bit before multiplexing is 1/1 kbps, or 0.001 s (1 ms).
2. The rate of the link is 4 times the rate of a connection, or 4 kbps.
3. The duration of each time slot is one-fourth of the duration of each bit before multiplexing, or 1/4 ms or 250 µs. Note that we can also calculate this from the data rate of the link, 4 kbps. The bit duration is the inverse of the data rate, or 1/4 kbps or 250 µs.
4. The duration of a frame is always the same as the duration of a unit before multiplexing, or 1 ms. We can also calculate this in another way. Each frame in this case has four time slots. So the duration of a frame is 4 times 250 µs, or 1 ms.

Interleaving

TDM can be visualized as two fast-rotating switches, one on the multiplexing side and the other on the demultiplexing side. The switches are synchronized and rotate at the same speed, but in opposite directions. On the multiplexing side, as the switch opens in front of a connection, that connection has the opportunity to send a unit onto the path. This process is called **interleaving.** On the demultiplexing side, as the switch opens in front of a connection, that connection has the opportunity to receive a unit from the path.

Figure 6.15 shows the interleaving process for the connection shown in Figure 6.13. In this figure, we assume that no switching is involved and that the data from the first connection at the multiplexer site go to the first connection at the demultiplexer. We discuss switching in Chapter 8.

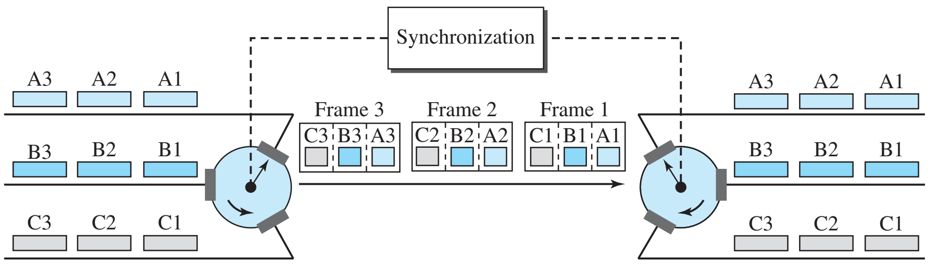
Example 6.8

Four channels are multiplexed using TDM. If each channel sends 100 bytes/s and we multiplex 1 byte per channel, show the frame traveling on the link, the size of the frame, the duration of a frame, the frame rate, and the bit rate for the link.

Solution

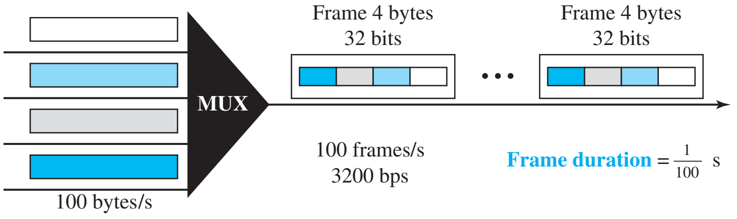
The multiplexer is shown in Figure 6.16. Each frame carries 1 byte from each channel; the size of each frame, therefore, is 4 bytes, or 32 bits. Because each channel is sending 100 bytes/s and a frame carries 1 byte from each channel, the frame rate must be 100 frames per second. The

Figure 6.15 *Interleaving*



duration of a frame is therefore 1/100 s. The link is carrying 100 frames per second, and since each frame contains 32 bits, the bit rate is 100 × 32, or 3200 bps. This is actually 4 times the bit rate of each channel, which is 100 × 8 = 800 bps.

Figure 6.16 *Example 6.8*



Example 6.9

A multiplexer combines four 100-kbps channels using a time slot of 2 bits. Show the output with four arbitrary inputs. What is the frame rate? What is the frame duration? What is the bit rate? What is the bit duration?

Solution

Figure 6.17 shows the output for four arbitrary inputs. The link carries 50,000 frames per second since each frame contains 2 bits per channel. The frame duration is therefore 1/50,000 s or 20 μs. The frame rate is 50,000 frames per second, and each frame carries 8 bits; the bit rate is 50,000 × 8 = 400,000 bits or 400 kbps. The bit duration is 1/400,000 s, or 2.5 μs. Note that the frame duration is 8 times the bit duration because each frame is carrying 8 bits.

Empty Slots

Synchronous TDM is not as efficient as it could be. If a source does not have data to send, the corresponding slot in the output frame is empty. Figure 6.18 shows a case in which one of the input lines has no data to send and one slot in another input line has discontinuous data.

The first output frame has three slots filled, the second frame has two slots filled, and the third frame has three slots filled. No frame is full. We learn in the next section

Figure 6.17 *Example 6.9*

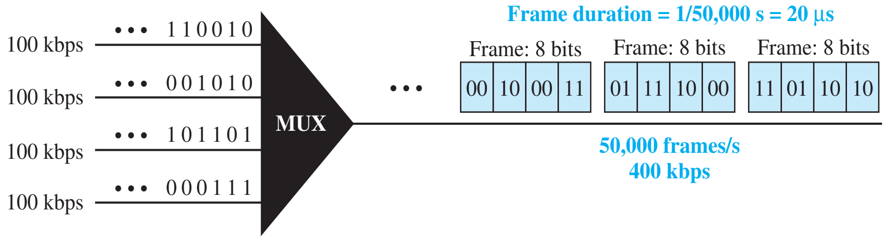
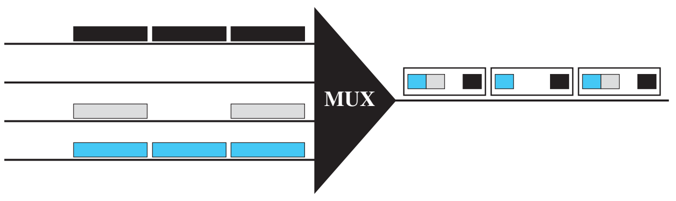


Figure 6.18 *Empty slots*



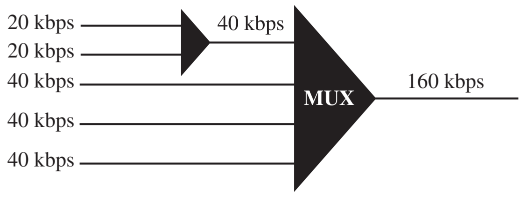
that statistical TDM can improve the efficiency by removing the empty slots from the frame.

Data Rate Management

One problem with TDM is how to handle a disparity in the input data rates. In all our discussion so far, we assumed that the data rates of all input lines were the same. However, if data rates are not the same, three strategies, or a combination of them, can be used. We call these three strategies **multilevel multiplexing, multiple-slot allocation,** and **pulse stuffing.**

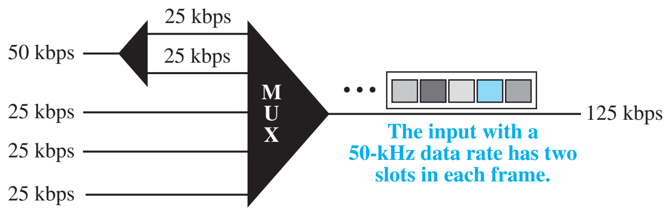
Multilevel Multiplexing Multilevel multiplexing is a technique used when the data rate of an input line is a multiple of others. For example, in Figure 6.19, we have two inputs of 20 kbps and three inputs of 40 kbps. The first two input lines can be multiplexed together to provide a data rate equal to the last three. A second level of multiplexing can create an output of 160 kbps.

Figure 6.19 *Multilevel multiplexing*



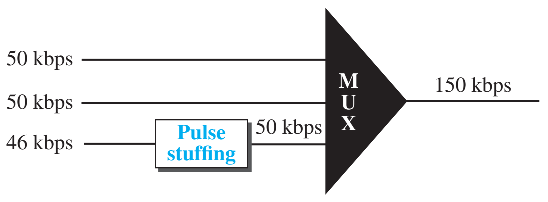
Multiple-Slot Allocation Sometimes it is more efficient to allot more than one slot in a frame to a single input line. For example, we might have an input line that has a data rate that is a multiple of another input. In Figure 6.20, the input line with a 50-kbps data rate can be given two slots in the output. We insert a demultiplexer in the line to make two inputs out of one.

Figure 6.20 *Multiple-slot multiplexing*


25 kbps
50 kbps
25 kbps
25 kbps
25 kbps
25 kbps
M U X
125 kbps
The input with a 50-kHz data rate has two slots in each frame.

Pulse Stuffing Sometimes the bit rates of sources are not multiple integers of each other. Therefore, neither of the above two techniques can be applied. One solution is to make the highest input data rate the dominant data rate and then add dummy bits to the input lines with lower rates. This will increase their rates. This technique is called ***pulse stuffing,*** *bit padding,* or *bit stuffing.* The idea is shown in Figure 6.21. The input with a data rate of 46 is pulse-stuffed to increase the rate to 50 kbps. Now multiplexing can take place.

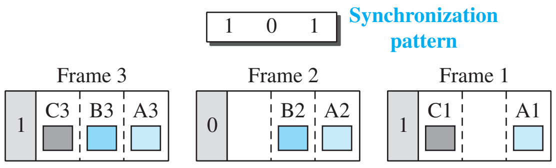
Figure 6.21 *Pulse stuffing*



Frame Synchronizing

The implementation of TDM is not as simple as that of FDM. Synchronization between the multiplexer and demultiplexer is a major issue. If the multiplexer and the demultiplexer are not synchronized, a bit belonging to one channel may be received by the wrong channel. For this reason, one or more synchronization bits are usually added to the beginning of each frame. These bits, called **framing bits,** follow a pattern, frame to frame, that allows the demultiplexer to synchronize with the incoming stream so that it can separate the time slots accurately. In most cases, this synchronization information consists of 1 bit per frame, alternating between 0 and 1, as shown in Figure 6.22.

Figure 6.22 *Framing bits*



Example 6.10

We have four sources, each creating 250 characters per second. If the interleaved unit is a character and 1 synchronizing bit is added to each frame, find (1) the data rate of each source, (2) the duration of each character in each source, (3) the frame rate, (4) the duration of each frame, (5) the number of bits in each frame, and (6) the data rate of the link.

Solution

We can answer the questions as follows:

1. The data rate of each source is $250 \times 8 = 2000$ bps = 2 kbps.
2. Each source sends 250 characters per second; therefore, the duration of a character is 1/250 s, or 4 ms.
3. Each frame has one character from each source, which means the link needs to send 250 frames per second to keep the transmission rate of each source.
4. The duration of each frame is 1/250 s, or 4 ms. Note that the duration of each frame is the same as the duration of each character coming from each source.
5. Each frame carries 4 characters and 1 extra synchronizing bit. This means that each frame is $4 \times 8 + 1 = 33$ bits.
6. The link sends 250 frames per second, and each frame contains 33 bits. This means that the data rate of the link is 250×33, or 8250 bps. Note that the bit rate of the link is greater than the combined bit rates of the four channels. If we add the bit rates of four channels, we get 8000 bps. Because 250 frames are traveling per second and each contains 1 extra bit for synchronizing, we need to add 250 to the sum to get 8250 bps.

Example 6.11

Two channels, one with a bit rate of 100 kbps and another with a bit rate of 200 kbps, are to be multiplexed. How this can be achieved? What is the frame rate? What is the frame duration? What is the bit rate of the link?

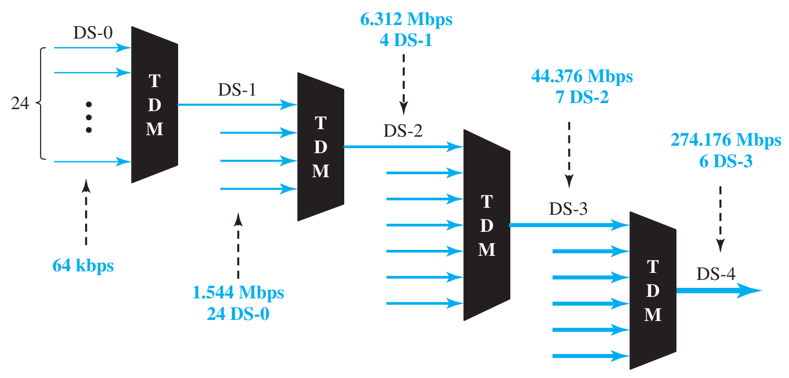
Solution

We can allocate one slot to the first channel and two slots to the second channel. Each frame carries 3 bits. The frame rate is 100,000 frames per second because it carries 1 bit from the first channel. The frame duration is 1/100,000 s, or 10 ms. The bit rate is 100,000 frames/s $\times$ 3 bits per frame, or 300 kbps. Note that because each frame carries 1 bit from the first channel, the bit rate for the first channel is preserved. The bit rate for the second channel is also preserved because each frame carries 2 bits from the second channel.

Digital Signal Service

Telephone companies implement TDM through a hierarchy of digital signals, called ***digital signal (DS) service*** or *digital hierarchy*. Figure 6.23 shows the data rates supported by each level.

Figure 6.23 *Digital hierarchy*



- **DS-0** is a single digital channel of 64 kbps.
- **DS-1** is a 1.544-Mbps service; 1.544 Mbps is 24 times 64 kbps plus 8 kbps of overhead. It can be used as a single service for 1.544-Mbps transmissions, or it can be used to multiplex 24 DS-0 channels or to carry any other combination desired by the user that can fit within its 1.544-Mbps capacity.
- **DS-2** is a 6.312-Mbps service; 6.312 Mbps is 96 times 64 kbps plus 168 kbps of overhead. It can be used as a single service for 6.312-Mbps transmissions; or it can be used to multiplex 4 DS-1 channels, 96 DS-0 channels, or a combination of these service types.
- **DS-3** is a 44.376-Mbps service; 44.376 Mbps is 672 times 64 kbps plus 1.368 Mbps of overhead. It can be used as a single service for 44.376-Mbps transmissions; or it can be used to multiplex 7 DS-2 channels, 28 DS-1 channels, 672 DS-0 channels, or a combination of these service types.
- **DS-4** is a 274.176-Mbps service; 274.176 is 4032 times 64 kbps plus 16.128 Mbps of overhead. It can be used to multiplex 6 DS-3 channels, 42 DS-2 channels, 168 DS-1 channels, 4032 DS-0 channels, or a combination of these service types.

T Lines

DS-0, DS-1, and so on are the names of services. To implement those services, the telephone companies use **T lines** (T-1 to T-4). These are lines with capacities precisely matched to the data rates of the DS-1 to DS-4 services (see Table 6.1). So far only T-1 and T-3 lines are commercially available.

Table 6.1 *DS and T line rates*

Service	*Line*	*Rate (Mbps)*	*Voice Channels*
DS-1	T-1	1.544	24
DS-2	T-2	6.312	96
DS-3	T-3	44.736	672
DS-4	T-4	274.176	4032

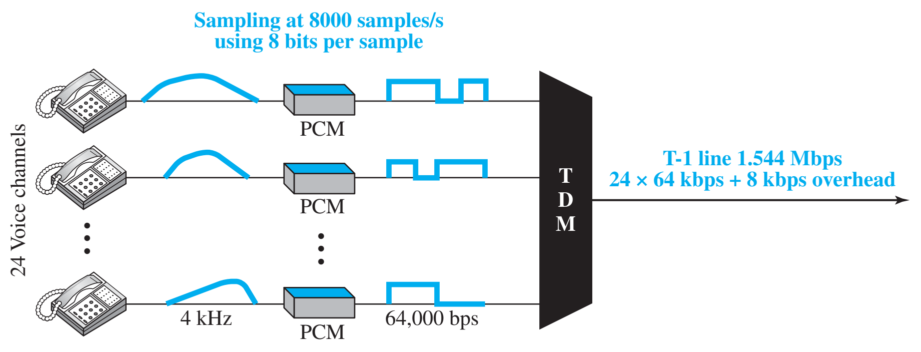
The T-1 line is used to implement DS-1; T-2 is used to implement DS-2; and so on. As you can see from Table 6.1, DS-0 is not actually offered as a service, but it has been defined as a basis for reference purposes.

T Lines for Analog Transmission

T lines are digital lines designed for the transmission of digital data, audio, or video. However, they also can be used for analog transmission (regular telephone connections), provided the analog signals are first sampled, then time-division multiplexed.

The possibility of using T lines as analog carriers opened up a new generation of services for the telephone companies. Earlier, when an organization wanted 24 separate telephone lines, it needed to run 24 twisted-pair cables from the company to the central exchange. (Remember those old movies showing a busy executive with 10 telephones lined up on his desk? Or the old office telephones with a big fat cable running from them? Those cables contained a bundle of separate lines.) Today, that same organization can combine the 24 lines into one T-1 line and run only the T-1 line to the exchange. Figure 6.24 shows how 24 voice channels can be multiplexed onto one T-1 line. (Refer to Chapter 4 for PCM encoding.)

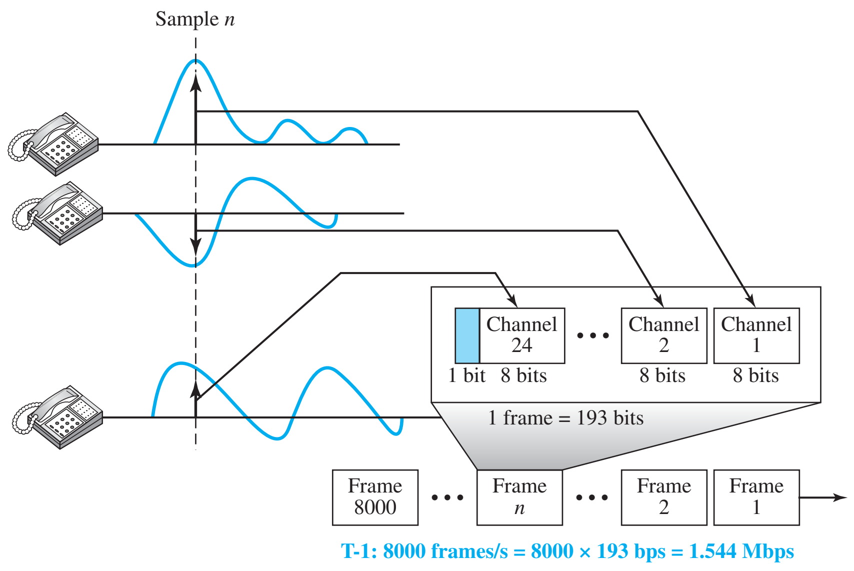
Figure 6.24 *T-1 line for multiplexing telephone lines*


Sampling at 8000 samples/s using 8 bits per sample
24 Voice channels
PCM
PCM
PCM
4 kHz
64,000 bps
TDM
T-1 line 1.544 Mbps
24 × 64 kbps + 8 kbps overhead

The T-1 Frame As noted above, DS-1 requires 8 kbps of overhead. To understand how this overhead is calculated, we must examine the format of a 24-voice-channel frame.

The frame used on a T-1 line is usually 193 bits divided into 24 slots of 8 bits each plus 1 extra bit for synchronization ($24 \times 8 + 1 = 193$); see Figure 6.25. In other words,

Figure 6.25 *T-1 frame structure*



each slot contains one signal segment from each channel; 24 segments are interleaved in one frame. If a T-1 line carries 8000 frames, the data rate is 1.544 Mbps ($193 \times 8000 = 1.544$ Mbps)—the capacity of the line.

E Lines

Europeans use a version of T lines called **E lines.** The two systems are conceptually identical, but their capacities differ. Table 6.2 shows the E lines and their capacities.

Table 6.2 *E line rates*

Line	*Rate (Mbps)*	*Voice Channels*
E-1	2.048	30
E-2	8.448	120
E-3	34.368	480
E-4	139.264	1920

More Synchronous TDM Applications

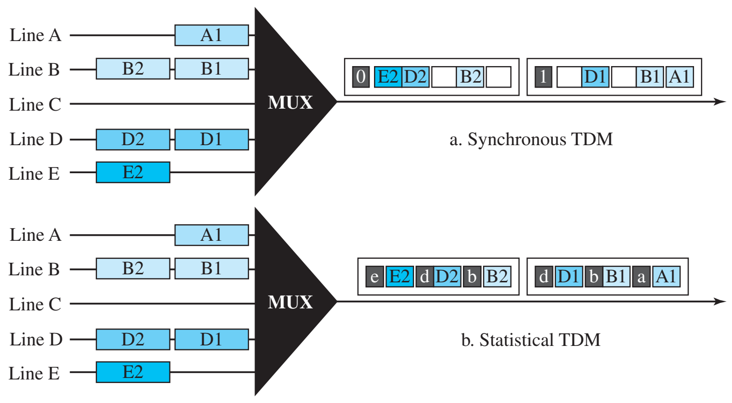
Some second-generation cellular telephone companies use synchronous TDM. For example, the digital version of cellular telephony divides the available bandwidth into 30-kHz bands. For each band, TDM is applied so that six users can share the band. This means that each 30-kHz band is now made of six time slots, and the digitized voice signals of the users are inserted in the slots. Using TDM, the number of telephone users in each area is now 6 times greater. We discuss second-generation cellular telephony in Chapter 16.

Statistical Time-Division Multiplexing

As we saw in the previous section, in synchronous TDM, each input has a reserved slot in the output frame. This can be inefficient if some input lines have no data to send. In statistical time-division multiplexing, slots are dynamically allocated to improve bandwidth efficiency. Only when an input line has a slot's worth of data to send is it given a slot in the output frame. In statistical multiplexing, the number of slots in each frame is less than the number of input lines. The multiplexer checks each input line in round-robin fashion; it allocates a slot for an input line if the line has data to send; otherwise, it skips the line and checks the next line.

Figure 6.26 shows a synchronous and a statistical TDM example. In the former, some slots are empty because the corresponding line does not have data to send. In the latter, however, no slot is left empty as long as there are data to be sent by any input line.

Figure 6.26 *TDM slot comparison*



Addressing

Figure 6.26 also shows a major difference between slots in synchronous TDM and statistical TDM. An output slot in synchronous TDM is totally occupied by data; in statistical TDM, a slot needs to carry data as well as the address of the destination. In synchronous TDM, there is no need for addressing; synchronization and preassigned relationships between the inputs and outputs serve as an address. We know, for example, that input 1 always goes to input 2. If the multiplexer and the demultiplexer are synchronized, this is guaranteed. In statistical multiplexing, there is no fixed relationship between the inputs and outputs because there are no preassigned or reserved slots. We need to include the address of the receiver inside each slot to show where it is to be delivered. The addressing in its simplest form can be n bits to define N different output lines with $n = \log_2 N$. For example, for eight different output lines, we need a 3-bit address.

Slot Size

Since a slot carries both data and an address in statistical TDM, the ratio of the data size to address size must be reasonable to make transmission efficient. For example, it would be inefficient to send 1 bit per slot as data when the address is 3 bits. This would mean an overhead of 300 percent. In statistical TDM, a block of data is usually many bytes while the address is just a few bytes.

No Synchronization Bit

There is another difference between synchronous and statistical TDM, but this time it is at the frame level. The frames in statistical TDM need not be synchronized, so we do not need synchronization bits.

Bandwidth

In statistical TDM, the capacity of the link is normally less than the sum of the capacities of each channel. The designers of statistical TDM define the capacity of the link based on the statistics of the load for each channel. If on average only x percent of the input slots are filled, the capacity of the link reflects this. Of course, during peak times, some slots need to wait.

6.2 SPREAD SPECTRUM

Multiplexing combines signals from several sources to achieve bandwidth efficiency; the available bandwidth of a link is divided between the sources. In **spread spectrum (SS),** we also combine signals from different sources to fit into a larger bandwidth, but our goals are somewhat different. Spread spectrum is designed to be used in wireless applications (LANs and WANs). In these types of applications, we have some concerns that outweigh bandwidth efficiency. In wireless applications, all stations use air (or a vacuum) as the medium for communication. Stations must be able to share this medium without interception by an eavesdropper and without being subject to jamming from a malicious intruder (in military operations, for example).

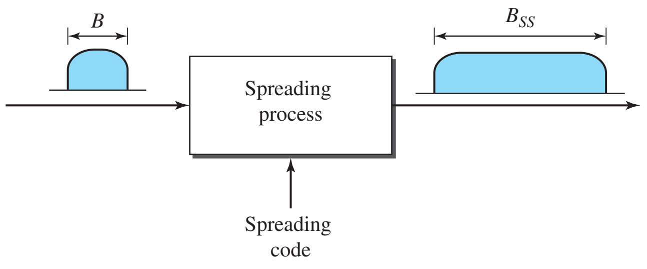
To achieve these goals, spread spectrum techniques add redundancy; they spread the original spectrum needed for each station. If the required bandwidth for each station is B, spread spectrum expands it to B_{ss}, such that $B_{ss} >> B$. The expanded bandwidth allows the source to wrap its message in a protective envelope for a more secure transmission. An analogy is the sending of a delicate, expensive gift. We can insert the gift in a special box to prevent it from being damaged during transportation, and we can use a superior delivery service to guarantee the safety of the package.

Figure 6.27 shows the idea of spread spectrum. Spread spectrum achieves its goals through two principles:

1. The bandwidth allocated to each station needs to be, by far, larger than what is needed. This allows redundancy.

2. The expanding of the original bandwidth B to the bandwidth B_{ss} must be done by a process that is independent of the original signal. In other words, the spreading process occurs after the signal is created by the source.

Figure 6.27 *Spread spectrum*



After the signal is created by the source, the spreading process uses a spreading code and spreads the bandwidth. The figure shows the original bandwidth B and the spread bandwidth B_{SS}. The spreading code is a series of numbers that look random, but are actually a pattern.

There are two techniques to spread the bandwidth: frequency hopping spread spectrum (FHSS) and direct sequence spread spectrum (DSSS).

6.2.1 Frequency Hopping Spread Spectrum

The **frequency hopping spread spectrum (FHSS)** technique uses M different carrier frequencies that are modulated by the source signal. At one moment, the signal modulates one carrier frequency; at the next moment, the signal modulates another carrier frequency. Although the modulation is done using one carrier frequency at a time, M frequencies are used in the long run. The bandwidth occupied by a source after spreading is $B_{FHSS} >> B$.

Figure 6.28 shows the general layout for FHSS. A **pseudorandom code generator,** called ***pseudorandom noise*** **(PN),** creates a k-bit pattern for every **hopping period** T_h. The frequency table uses the pattern to find the frequency to be used for this hopping period and passes it to the frequency synthesizer. The frequency synthesizer creates a carrier signal of that frequency, and the source signal modulates the carrier signal.

Suppose we have decided to have eight hopping frequencies. This is extremely low for real applications and is just for illustration. In this case, M is 8 and k is 3. The pseudorandom code generator will create eight different 3-bit patterns. These are mapped to eight different frequencies in the frequency table (see Figure 6.29).

The pattern for this station is 101, 111, 001, 000, 010, 011, 100. Note that the pattern is pseudorandom; it is repeated after eight hoppings. This means that at hopping period 1, the pattern is 101. The frequency selected is 700 kHz; the source signal modulates this carrier frequency. The second k-bit pattern selected is 111, which selects the 900-kHz carrier; the eighth pattern is 100, and the frequency is 600 kHz. After eight hoppings, the pattern repeats, starting from 101 again. Figure 6.30 shows how the signal

Figure 6.28 *Frequency hopping spread spectrum (FHSS)*

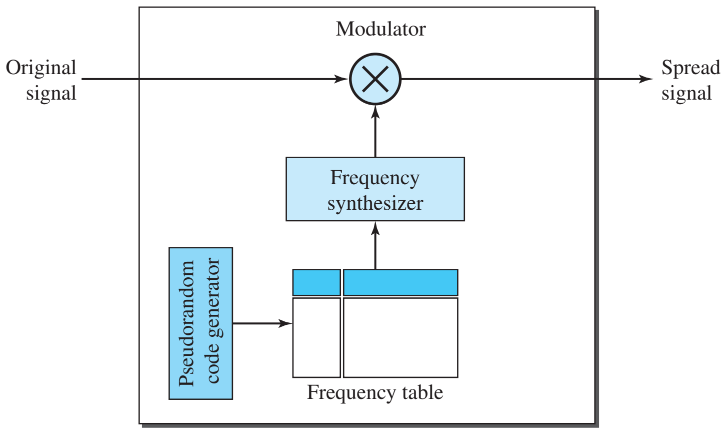
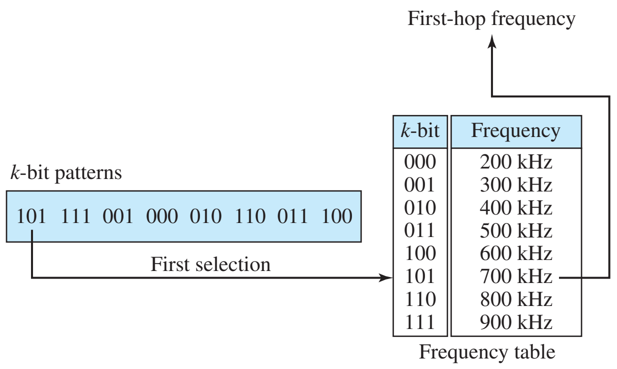


Figure 6.29 *Frequency selection in FHSS*



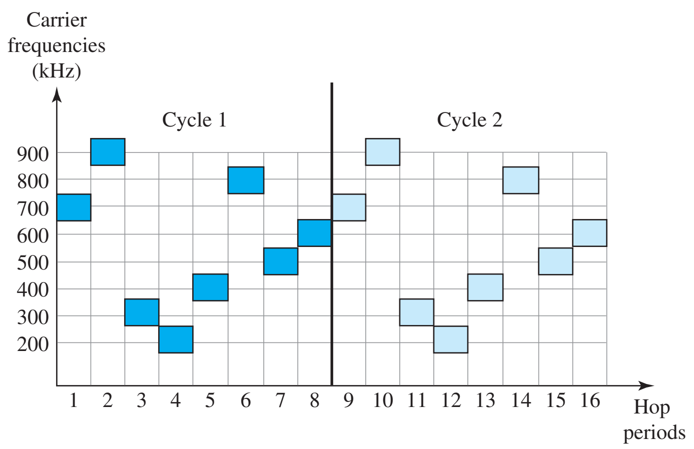
hops around from carrier to carrier. We assume the required bandwidth of the original signal is 100 kHz.

It can be shown that this scheme can accomplish the previously mentioned goals. If there are many *k*-bit patterns and the hopping period is short, a sender and receiver can have privacy. If an intruder tries to intercept the transmitted signal, she can only access a small piece of data because she does not know the spreading sequence to quickly adapt herself to the next hop. The scheme also has an antijamming effect. A malicious sender may be able to send noise to jam the signal for one hopping period (randomly), but not for the whole period.

Bandwidth Sharing

If the number of hopping frequencies is M, we can multiplex M channels into one by using the same B_{ss} bandwidth. This is possible because a station uses just one frequency in each hopping period; $M - 1$ other frequencies can be used by $M - 1$ other stations. In

Figure 6.30 *FHSS cycles*



other words, M different stations can use the same B_{ss} if an appropriate modulation technique such as multiple FSK (MFSK) is used. FHSS is similar to FDM, as shown in Figure 6.31.

Figure 6.31 *Bandwidth sharing*

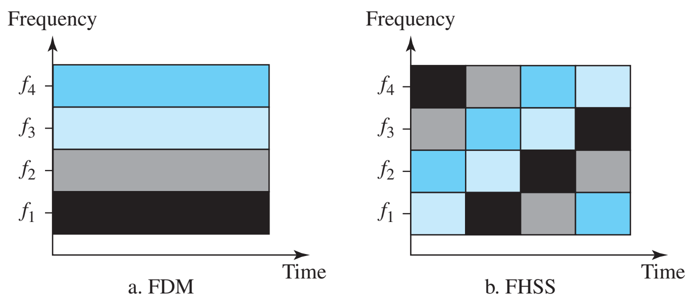
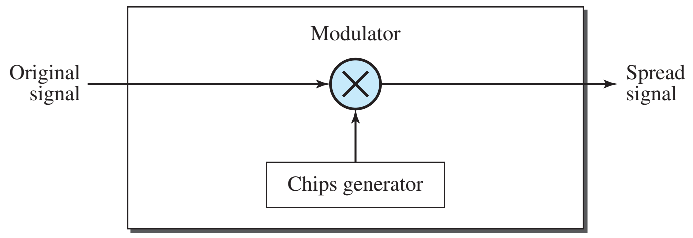


Figure 6.31 shows an example of four channels using FDM and four channels using FHSS. In FDM, each station uses $1/M$ of the bandwidth, but the allocation is fixed; in FHSS, each station uses $1/M$ of the bandwidth, but the allocation changes hop to hop.

6.2.2 Direct Sequence Spread Spectrum

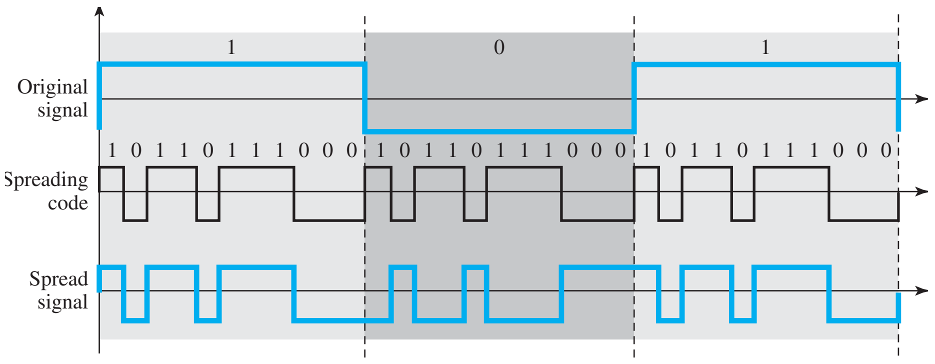
The **direct sequence spread spectrum (DSSS)** technique also expands the bandwidth of the original signal, but the process is different. In DSSS, we replace each data bit with n bits using a spreading code. In other words, each bit is assigned a code of n bits, called ***chips,*** where the chip rate is n times that of the data bit. Figure 6.32 shows the concept of DSSS.

Figure 6.32 *DSSS*



As an example, let us consider the sequence used in a wireless LAN, the famous **Barker sequence,** where *n* is 11. We assume that the original signal and the chips in the chip generator use polar NRZ encoding. Figure 6.33 shows the chips and the result of multiplying the original data by the chips to get the spread signal.

Figure 6.33 *DSSS example*



In Figure 6.33, the spreading code is 11 chips having the pattern 10110111000 (in this case). If the original signal rate is *N*, the rate of the spread signal is 11*N*. This means that the required bandwidth for the spread signal is 11 times larger than the bandwidth of the original signal. The spread signal can provide privacy if the intruder does not know the code. It can also provide immunity against interference if each station uses a different code.

Bandwidth Sharing

Can we share a bandwidth in DSSS as we did in FHSS? The answer is no and yes. If we use a spreading code that spreads signals (from different stations) that cannot be combined and separated, we cannot share a bandwidth. For example, as we will see in Chapter 15, some wireless LANs use DSSS and the spread bandwidth cannot be shared. However, if we use a special type of sequence code that allows the combining and separating of spread signals, we can share the bandwidth. As we will see in

Chapter 16, a special spreading code allows us to use DSSS in cellular telephony and share a bandwidth among several users.

6.3 END-CHAPTER MATERIALS

6.3.1 Recommended Reading

For more details about subjects discussed in this chapter, we recommend the following books. The items in brackets [. . .] refer to the reference list at the end of the text.

Books

Multiplexing is discussed in [Pea92]. [Cou01] gives excellent coverage of TDM and FDM. More advanced materials can be found in [Ber96]. Multiplexing is discussed in [Sta04]. A good coverage of spread spectrum can be found in [Cou01] and [Sta04].

6.3.2 Key Terms

analog hierarchy
Barker sequence
channel
chip
demultiplexer (DEMUX)
dense WDM (DWDM)
digital signal (DS) service
direct sequence spread spectrum (DSSS)
E line
framing bit
frequency hopping spread spectrum (FHSS)
frequency-division multiplexing (FDM)
group
guard band
hopping period
interleaving
jumbo group
link
master group
multilevel multiplexing
multiple-slot allocation
multiplexer (MUX)
multiplexing
pseudorandom code generator
pseudorandom noise (PN)
pulse stuffing
spread spectrum (SS)
statistical TDM
supergroup
synchronous TDM
T line
time-division multiplexing (TDM)
wavelength-division multiplexing (WDM)

6.3.3 Summary

Bandwidth utilization is the use of available bandwidth to achieve specific goals. Efficiency can be achieved by using multiplexing; privacy and antijamming can be achieved by using spreading.

Multiplexing is the set of techniques that allow the simultaneous transmission of multiple signals across a single data link. In a multiplexed system, n lines share the bandwidth of one link. The word *link* refers to the physical path. The word *channel* refers to the portion of a link that carries a transmission. There are three basic multiplexing techniques: frequency-division multiplexing, wavelength-division multiplexing, and time-division multiplexing. The first two are techniques designed for analog signals, the third, for digital signals. Frequency-division multiplexing (FDM) is an analog technique that can be applied when the bandwidth of a link (in hertz) is greater than the combined bandwidths of the signals to be transmitted. Wavelength-division multiplexing (WDM) is designed to use the high bandwidth capability of fiber-optic cable. WDM is an analog multiplexing technique to combine optical signals. Time-division multiplexing (TDM) is a digital process that allows several connections to share the high bandwidth of a link. TDM is a digital multiplexing technique for combining several low-rate channels into one high-rate one. We can divide TDM into two different schemes: synchronous or statistical. In synchronous TDM, each input connection has an allotment in the output even if it is not sending data. In statistical TDM, slots are dynamically allocated to improve bandwidth efficiency.

In spread spectrum (SS), we combine signals from different sources to fit into a larger bandwidth. Spread spectrum is designed to be used in wireless applications in which stations must be able to share the medium without interception by an eavesdropper and without being subject to jamming from a malicious intruder. The frequency hopping spread spectrum (FHSS) technique uses M different carrier frequencies that are modulated by the source signal. At one moment, the signal modulates one carrier frequency; at the next moment, the signal modulates another carrier frequency. The direct sequence spread spectrum (DSSS) technique expands the bandwidth of a signal by replacing each data bit with *n* bits using a spreading code. In other words, each bit is assigned a code of *n* bits, called chips.

6.4 PRACTICE SET

6.4.1 Quizzes

A set of interactive quizzes for this chapter can be found on the book website. It is strongly recommended that the student take the quizzes to check his/her understanding of the materials before continuing with the practice set.

6.4.2 Questions

Q6-1. Describe the goals of multiplexing.

Q6-2. List three main multiplexing techniques mentioned in this chapter.

Q6-3. Distinguish between a link and a channel in multiplexing.

Q6-4. Which of the three multiplexing techniques is (are) used to combine analog signals? Which of the three multiplexing techniques is (are) used to combine digital signals?

Q6-5. Define the analog hierarchy used by telephone companies and list different levels of the hierarchy.

Q6-6. Define the digital hierarchy used by telephone companies and list different levels of the hierarchy.

Q6-7. Which of the three multiplexing techniques is common for fiber-optic links? Explain the reason.

Q6-8. Distinguish between multilevel TDM, multiple-slot TDM, and pulse-stuffed TDM.

Q6-9. Distinguish between synchronous and statistical TDM.

Q6-10. Define spread spectrum and its goal. List the two spread spectrum techniques discussed in this chapter.

Q6-11. Define FHSS and explain how it achieves bandwidth spreading.

Q6-12. Define DSSS and explain how it achieves bandwidth spreading.

6.4.3 Problems

P6-1. Assume that a voice channel occupies a bandwidth of 4 kHz. We need to multiplex 10 voice channels with guard bands of 500 Hz using FDM. Calculate the required bandwidth.

P6-2. We need to transmit 100 digitized voice channels using a passband channel of 20 KHz. What should be the ratio of bits/Hz if we use no guard band?

P6-3. In the analog hierarchy of Figure 6.9, find the overhead (extra bandwidth for guard band or control) in each hierarchy level (group, supergroup, master group, and jumbo group).

P6-4. We need to use synchronous TDM and combine 20 digital sources, each of 100 Kbps. Each output slot carries 1 bit from each digital source, but one extra bit is added to each frame for synchronization. Answer the following questions:

a. What is the size of an output frame in bits?

b. What is the output frame rate?

c. What is the duration of an output frame?

d. What is the output data rate?

e. What is the efficiency of the system (ratio of useful bits to the total bits)?

P6-5. Repeat Problem 6-4 if each output slot carries 2 bits from each source.

P6-6. We have 14 sources, each creating 500 8-bit characters per second. Since only some of these sources are active at any moment, we use statistical TDM to combine these sources using character interleaving. Each frame carries 6 slots at a time, but we need to add 4-bit addresses to each slot. Answer the following questions:

a. What is the size of an output frame in bits?

b. What is the output frame rate?

c. What is the duration of an output frame?

d. What is the output data rate?

P6-7. Ten sources, six with a bit rate of 200 kbps and four with a bit rate of 400 kbps, are to be combined using multilevel TDM with no synchronizing bits. Answer the following questions about the final stage of the multiplexing:

a. What is the size of a frame in bits?

b. What is the frame rate?

c. What is the duration of a frame?

d. What is the data rate?

P6-8. Four channels, two with a bit rate of 200 kbps and two with a bit rate of 150 kbps, are to be multiplexed using multiple-slot TDM with no synchronization bits. Answer the following questions:

a. What is the size of a frame in bits?

b. What is the frame rate?

c. What is the duration of a frame?

d. What is the data rate?

P6-9. Two channels, one with a bit rate of 190 kbps and another with a bit rate of 180 kbps, are to be multiplexed using pulse-stuffing TDM with no synchronization bits. Answer the following questions:

a. What is the size of a frame in bits?

b. What is the frame rate?

c. What is the duration of a frame?

d. What is the data rate?

P6-10. Answer the following questions about a T-1 line:

a. What is the duration of a frame?

b. What is the overhead (number of extra bits per second)?

P6-11. Show the contents of the five output frames for a synchronous TDM multiplexer that combines four sources sending the following characters. Note that the characters are sent in the same order that they are typed. The third source is silent.

a. Source 1 message: HELLO

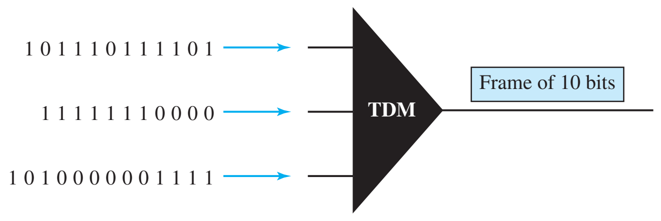
b. Source 2 message: HI

c. Source 3 message:

d. Source 4 message: BYE

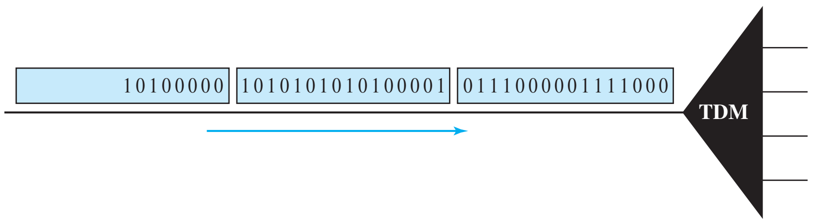
P6-12. Figure 6.34 shows a multiplexer in a synchronous TDM system. Each output slot is only 10 bits long (3 bits taken from each input plus 1 framing bit). What is the output stream? The bits arrive at the multiplexer as shown by the arrows.

Figure 6.34 *Problem P6-12*



P6-13. Figure 6.35 shows a demultiplexer in a synchronous TDM. If the input slot is 16 bits long (no framing bits), what is the bit stream in each output? The bits arrive at the demultiplexer as shown by the arrows.

Figure 6.35 *Problem P6-13*



P6-14. Answer the following questions about the digital hierarchy in Figure 6.23:

- **a.** What is the overhead (number of extra bits) in the DS-1 service?
- **b.** What is the overhead (number of extra bits) in the DS-2 service?
- **c.** What is the overhead (number of extra bits) in the DS-3 service?
- **d.** What is the overhead (number of extra bits) in the DS-4 service?

P6-15. What is the minimum number of bits in a PN sequence if we use FHSS with a channel bandwidth of $B = 4$ KHz and $B_{SS} = 100$ KHz?

P6-16. An FHSS system uses a 4-bit PN sequence. If the bit rate of the PN is 64 bits per second, answer the following questions:

- **a.** What is the total number of possible channels?
- **b.** What is the time needed to finish a complete cycle of PN?

P6-17. A pseudorandom number generator uses the following formula to create a random series:

$$N_{i+1} = (5 + 7N_i) \bmod 17 - 1$$

In which N_i defines the current random number and N_{i+1} defines the next random number. The term *mod* means the value of the remainder when dividing $(5 + 7N_i)$ by 17. Show the sequence created by this generator to be used for spread spectrum.

P6-18. We have a digital medium with a data rate of 10 Mbps. How many 64-kbps voice channels can be carried by this medium if we use DSSS with the Barker sequence?

6.5 SIMULATION EXPERIMENTS

6.5.1 Applets

We have created some Java applets to show some of the main concepts discussed in this chapter. It is strongly recommended that the students activate these applets on the book website and carefully examine the protocols in action.

CHAPTER 8

Switching

Switching is a topic that can be discussed at several layers. We have switching at the physical layer, at the data-link layer, at the network layer, and even logically at the application layer (message switching). We have decided to discuss the general idea behind switching in this chapter, the last chapter related to the physical layer. We particularly discuss circuit-switching, which occurs at the physical layer. We introduce the idea of packet-switching, which occurs at the data-link and network layers, but we postpone the details of these topics until the appropriate chapters. Finally, we talk about the physical structures of the switches and routers.

This chapter is divided into four sections:

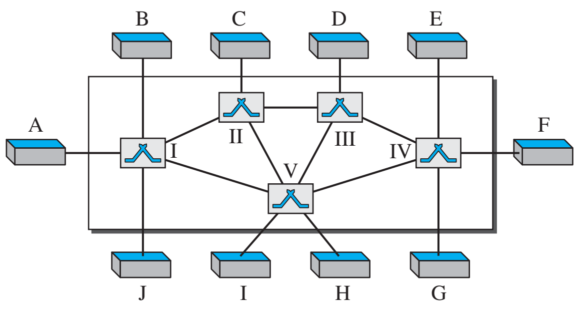
- The first section introduces switching. It mentions three methods of switching: circuit switching, packet switching, and message switching. The section then defines the switching methods that can occur in some layers of the Internet model.
- The second section discusses circuit-switched networks. It first defines three phases in these types of networks. It then describes the efficiency of these networks. The section also discusses the delay in circuit-switched networks.
- The third section briefly discusses packet-switched networks. It first describes datagram networks, listing their characteristics and advantages. The section then describes virtual circuit networks, explaining their features and operations. We will discuss packet-switched networks in more detail in Chapter 18.
- The last section discusses the structure of a switch. It first describes the structure of a circuit switch. It then explains the structure of a packet switch.

8.1 INTRODUCTION

A network is a set of connected devices. Whenever we have multiple devices, we have the problem of how to connect them to make one-to-one communication possible. One solution is to make a point-to-point connection between each pair of devices (a mesh topology) or between a central device and every other device (a star topology). These methods, however, are impractical and wasteful when applied to very large networks. The number and length of the links require too much infrastructure to be cost-efficient, and the majority of those links would be idle most of the time. Other topologies employing multipoint connections, such as a bus, are ruled out because the distances between devices and the total number of devices increase beyond the capacities of the media and equipment.

A better solution is **switching.** A switched network consists of a series of interlinked nodes, called ***switches*.** Switches are devices capable of creating temporary connections between two or more devices linked to the switch. In a switched network, some of these nodes are connected to the end systems (computers or telephones, for example). Others are used only for routing. Figure 8.1 shows a switched network.

Figure 8.1 *Switched network*

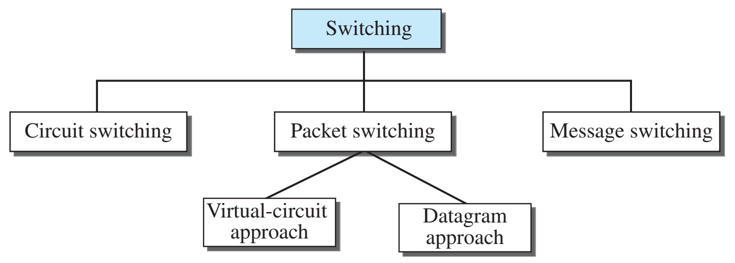


The **end systems** (communicating devices) are labeled A, B, C, D, and so on, and the switches are labeled I, II, III, IV, and V. Each switch is connected to multiple links.

8.1.1 Three Methods of Switching

Traditionally, three methods of switching have been discussed: **circuit switching**, **packet switching**, and **message switching**. The first two are commonly used today. The third has been phased out in general communications but still has networking applications. Packet switching can further be divided into two subcategories—virtual-circuit approach and datagram approach—as shown in Figure 8.2. In this chapter, we discuss only circuit switching and packet switching; message switching is more conceptual than practical.

Figure 8.2 *Taxonomy of switched networks*



8.1.2 Switching and TCP/IP Layers

Switching can happen at several layers of the TCP/IP protocol suite.

Switching at Physical Layer

At the physical layer, we can have only circuit switching. There are no packets exchanged at the physical layer. The switches at the physical layer allow signals to travel in one path or another.

Switching at Data-Link Layer

At the data-link layer, we can have packet switching. However, the term *packet* in this case means *frames* or *cells*. Packet switching at the data-link layer is normally done using a virtual-circuit approach.

Switching at Network Layer

At the network layer, we can have packet switching. In this case, either a virtual-circuit approach or a datagram approach can be used. Currently the Internet uses a datagram approach, as we see in Chapter 18, but the tendency is to move to a virtual-circuit approach.

Switching at Application Layer

At the application layer, we can have only message switching. The communication at the application layer occurs by exchanging messages. Conceptually, we can say that communication using e-mail is a kind of message-switched communication, but we do not see any network that actually can be called a message-switched network.

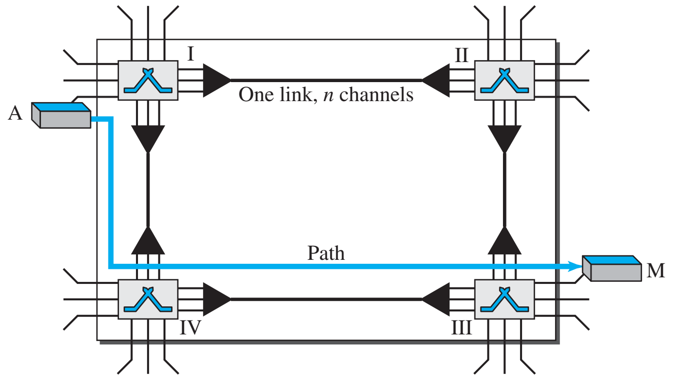
8.2 CIRCUIT-SWITCHED NETWORKS

A **circuit-switched network** consists of a set of switches connected by physical links. A connection between two stations is a dedicated path made of one or more links. However, each connection uses only one dedicated channel on each link. Each link is normally divided into *n* channels by using FDM or TDM, as discussed in Chapter 6.

A circuit-switched network is made of a set of switches connected by physical links, in which each link is divided into *n* channels.

Figure 8.3 shows a trivial circuit-switched network with four switches and four links. Each link is divided into *n* (*n* is 3 in the figure) channels by using FDM or TDM.

Figure 8.3 *A trivial circuit-switched network*



We have explicitly shown the multiplexing symbols to emphasize the division of the link into channels even though multiplexing can be implicitly included in the switch fabric.

The end systems, such as computers or telephones, are directly connected to a switch. We have shown only two end systems for simplicity. When end system A needs to communicate with end system M, system A needs to request a connection to M that must be accepted by all switches as well as by M itself. This is called the **setup phase;** a circuit (channel) is reserved on each link, and the combination of circuits or channels defines the dedicated path. After the dedicated path made of connected circuits (channels) is established, the **data-transfer phase** can take place. After all data have been transferred, the circuits are torn down.

We need to emphasize several points here:

- ❑ Circuit switching takes place at the physical layer.
- ❑ Before starting communication, the stations must make a reservation for the resources to be used during the communication. These resources, such as channels (bandwidth in FDM and time slots in TDM), switch buffers, switch processing time, and switch input/output ports, must remain dedicated during the entire duration of data transfer until the **teardown phase.**
- ❑ Data transferred between the two stations are not packetized (physical layer transfer of the signal). The data are a continuous flow sent by the source station and received by the destination station, although there may be periods of silence.

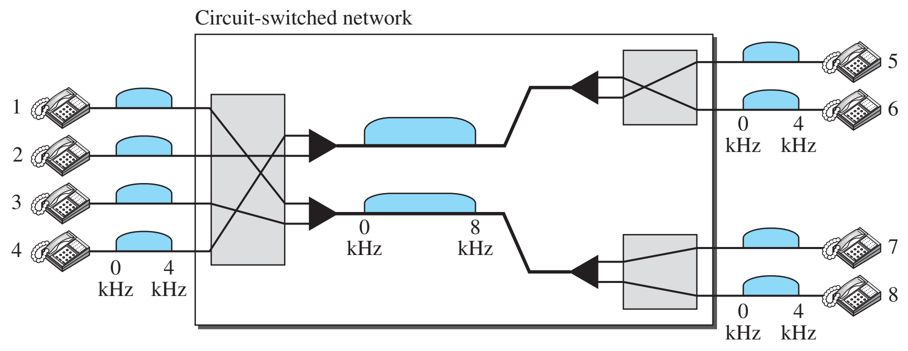
- There is no addressing involved during data transfer. The switches route the data based on their occupied band (FDM) or time slot (TDM). Of course, there is end-to-end addressing used during the setup phase, as we will see shortly.

In circuit switching, the resources need to be reserved during the setup phase; the resources remain dedicated for the entire duration of data transfer until the teardown phase.

Example 8.1

As a trivial example, let us use a circuit-switched network to connect eight telephones in a small area. Communication is through 4-kHz voice channels. We assume that each link uses FDM to connect a maximum of two voice channels. The bandwidth of each link is then 8 kHz. Figure 8.4 shows the situation. Telephone 1 is connected to telephone 7; 2 to 5; 3 to 8; and 4 to 6. Of course the situation may change when new connections are made. The switch controls the connections.

Figure 8.4 *Circuit-switched network used in Example 8.1*



Example 8.2

As another example, consider a circuit-switched network that connects computers in two remote offices of a private company. The offices are connected using a T-1 line leased from a communication service provider. There are two 4×8 (4 inputs and 8 outputs) switches in this network. For each switch, four output ports are folded into the input ports to allow communication between computers in the same office. Four other output ports allow communication between the two offices. Figure 8.5 shows the situation.

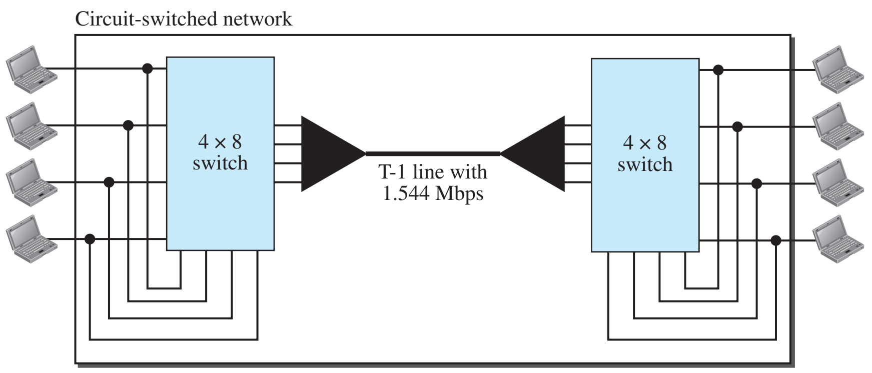
8.2.1 Three Phases

The actual communication in a circuit-switched network requires three phases: connection setup, data transfer, and connection teardown.

Setup Phase

Before the two parties (or multiple parties in a conference call) can communicate, a dedicated circuit (combination of channels in links) needs to be established. The end systems are normally connected through dedicated lines to the switches, so connection setup

Figure 8.5 *Circuit-switched network used in Example 8.2*



means creating dedicated channels between the switches. For example, in Figure 8.3, when system A needs to connect to system M, it sends a setup request that includes the address of system M, to switch I. Switch I finds a channel between itself and switch IV that can be dedicated for this purpose. Switch I then sends the request to switch IV, which finds a dedicated channel between itself and switch III. Switch III informs system M of system A's intention at this time.

In the next step to making a connection, an acknowledgment from system M needs to be sent in the opposite direction to system A. Only after system A receives this acknowledgment is the connection established.

Note that end-to-end addressing is required for creating a connection between the two end systems. These can be, for example, the addresses of the computers assigned by the administrator in a TDM network, or telephone numbers in an FDM network.

Data-Transfer Phase

After the establishment of the dedicated circuit (channels), the two parties can transfer data.

Teardown Phase

When one of the parties needs to disconnect, a signal is sent to each switch to release the resources.

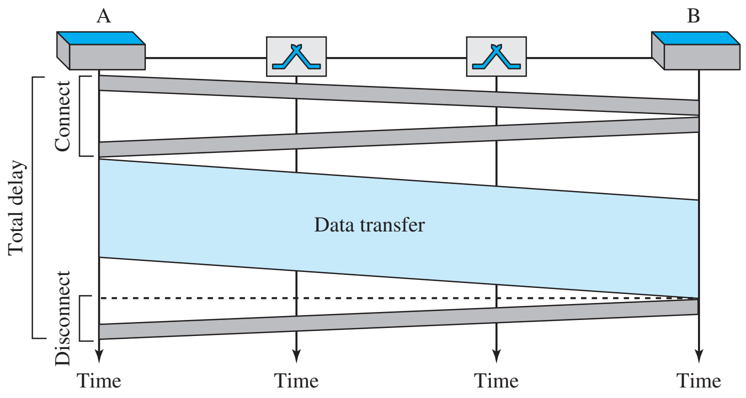
8.2.2 Efficiency

It can be argued that circuit-switched networks are not as efficient as the other two types of networks because resources are allocated during the entire duration of the connection. These resources are unavailable to other connections. In a telephone network, people normally terminate the communication when they have finished their conversation. However, in computer networks, a computer can be connected to another computer even if there is no activity for a long time. In this case, allowing resources to be dedicated means that other connections are deprived.

8.2.3 Delay

Although a circuit-switched network normally has low efficiency, the delay in this type of network is minimal. During data transfer the data are not delayed at each switch; the resources are allocated for the duration of the connection. Figure 8.6 shows the idea of delay in a circuit-switched network when only two switches are involved.

Figure 8.6 *Delay in a circuit-switched network*



As Figure 8.6 shows, there is no waiting time at each switch. The total delay is due to the time needed to create the connection, transfer data, and disconnect the circuit. The delay caused by the setup is the sum of four parts: the propagation time of the source computer request (slope of the first gray box), the request signal transfer time (height of the first gray box), the propagation time of the acknowledgment from the destination computer (slope of the second gray box), and the signal transfer time of the acknowledgment (height of the second gray box). The delay due to data transfer is the sum of two parts: the propagation time (slope of the colored box) and data transfer time (height of the colored box), which can be very long. The third box shows the time needed to tear down the circuit. We have shown the case in which the receiver requests disconnection, which creates the maximum delay.

8.3 PACKET SWITCHING

In data communications, we need to send messages from one end system to another. If the message is going to pass through a **packet-switched network**, it needs to be divided into packets of fixed or variable size. The size of the packet is determined by the network and the governing protocol.

In packet switching, there is no resource allocation for a packet. This means that there is no reserved bandwidth on the links, and there is no scheduled processing time for each packet. Resources are allocated on demand. The allocation is done on a first-come, first-served basis. When a switch receives a packet, no matter what the source or destination is, the packet must wait if there are other packets being processed. As with other systems in our daily life, this lack of reservation may create delay. For example, if we do not have a reservation at a restaurant, we might have to wait.

In a packet-switched network, there is no resource reservation; resources are allocated on demand.

We can have two types of packet-switched networks: datagram networks and virtual-circuit networks.

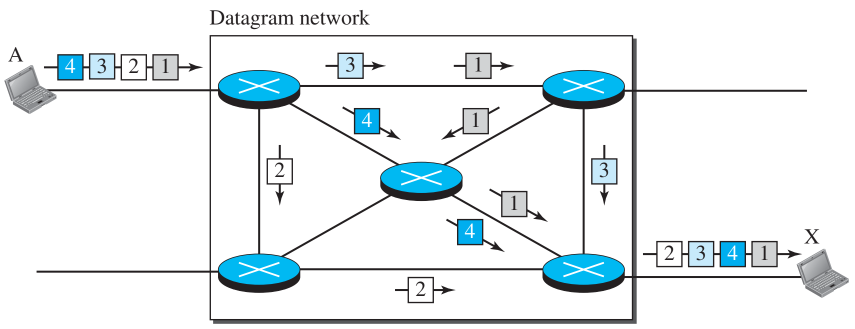
8.3.1 Datagram Networks

In a **datagram network**, each packet is treated independently of all others. Even if a packet is part of a multipacket transmission, the network treats it as though it existed alone. Packets in this approach are referred to as ***datagrams***.

Datagram switching is normally done at the network layer. We briefly discuss datagram networks here as a comparison with circuit-switched and virtual-circuit-switched networks. In Chapter 18 of this text, we go into greater detail.

Figure 8.7 shows how the datagram approach is used to deliver four packets from station A to station X. The switches in a datagram network are traditionally referred to as routers. That is why we use a different symbol for the switches in the figure.

Figure 8.7 *A datagram network with four switches (routers)*



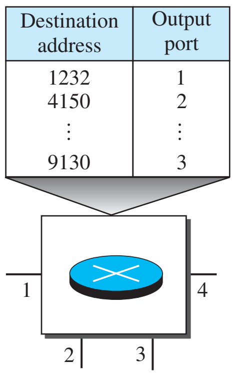
In this example, all four packets (or datagrams) belong to the same message, but may travel different paths to reach their destination. This is so because the links may be involved in carrying packets from other sources and do not have the necessary bandwidth available to carry all the packets from A to X. This approach can cause the datagrams of a transmission to arrive at their destination out of order with different delays between the packets. Packets may also be lost or dropped because of a lack of resources. In most protocols, it is the responsibility of an upper-layer protocol to reorder the datagrams or ask for lost datagrams before passing them on to the application.

The datagram networks are sometimes referred to as *connectionless networks*. The term *connectionless* here means that the switch (packet switch) does not keep information about the connection state. There are no setup or teardown phases. Each packet is treated the same by a switch regardless of its source or destination.

Routing Table

If there are no setup or teardown phases, how are the packets routed to their destinations in a datagram network? In this type of network, each switch (or packet switch) has a routing table which is based on the destination address. The routing tables are dynamic and are updated periodically. The destination addresses and the corresponding forwarding output ports are recorded in the tables. This is different from the table of a circuit-switched network (discussed later) in which each entry is created when the setup phase is completed and deleted when the teardown phase is over. Figure 8.8 shows the routing table for a switch.

Figure 8.8 *Routing table in a datagram network*



Destination address	Output port
1232	1
4150	2
⋮	⋮
9130	3

A switch in a datagram network uses a routing table that is based on the destination address.

Destination Address

Every packet in a datagram network carries a header that contains, among other information, the destination address of the packet. When the switch receives the packet, this destination address is examined; the routing table is consulted to find the corresponding port through which the packet should be forwarded. This address, unlike the address in a virtual-circuit network, remains the same during the entire journey of the packet.

The destination address in the header of a packet in a datagram network remains the same during the entire journey of the packet.

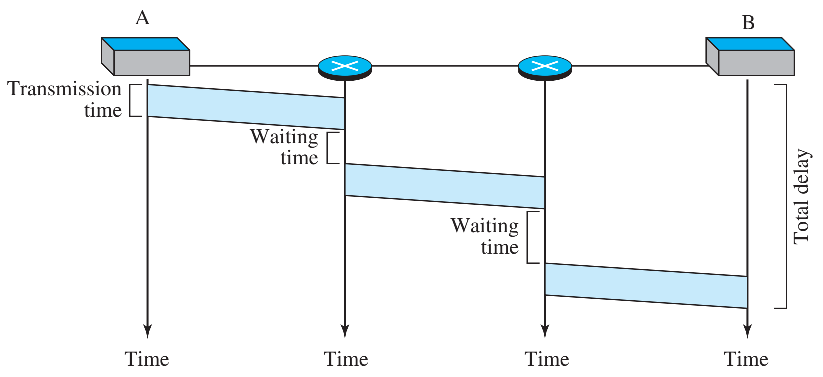
Efficiency

The efficiency of a datagram network is better than that of a circuit-switched network; resources are allocated only when there are packets to be transferred. If a source sends a packet and there is a delay of a few minutes before another packet can be sent, the resources can be reallocated during these minutes for other packets from other sources.

Delay

There may be greater delay in a datagram network than in a virtual-circuit network. Although there are no setup and teardown phases, each packet may experience a wait at a switch before it is forwarded. In addition, since not all packets in a message necessarily travel through the same switches, the delay is not uniform for the packets of a message. Figure 8.9 gives an example of delay in a datagram network for one packet.

Figure 8.9 *Delay in a datagram network*



The packet travels through two switches. There are three transmission times ($3T$), three propagation delays (slopes 3τ of the lines), and two waiting times ($w_1 + w_2$). We ignore the processing time in each switch. The total delay is

$$\textbf{Total delay} = 3T + 3\tau + w_1 + w_2$$

8.3.2 Virtual-Circuit Networks

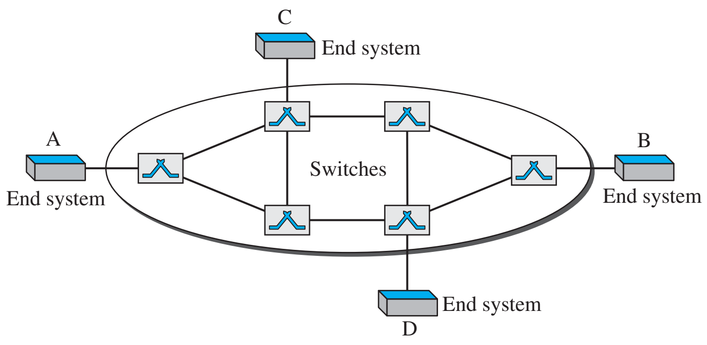
A **virtual-circuit network** is a cross between a circuit-switched network and a datagram network. It has some characteristics of both.

1. As in a circuit-switched network, there are setup and teardown phases in addition to the data transfer phase.
2. Resources can be allocated during the setup phase, as in a circuit-switched network, or on demand, as in a datagram network.
3. As in a datagram network, data are packetized and each packet carries an address in the header. However, the address in the header has local jurisdiction (it defines what the next switch should be and the channel on which the packet is being carried), not end-to-end jurisdiction. The reader may ask how the intermediate switches know where to send the packet if there is no final destination address carried by a packet. The answer will be clear when we discuss virtual-circuit identifiers in the next section.
4. As in a circuit-switched network, all packets follow the same path established during the connection.

5. A virtual-circuit network is normally implemented in the data-link layer, while a circuit-switched network is implemented in the physical layer and a datagram network in the network layer. But this may change in the future.

Figure 8.10 is an example of a virtual-circuit network. The network has switches that allow traffic from sources to destinations. A source or destination can be a computer, packet switch, bridge, or any other device that connects other networks.

Figure 8.10 *Virtual-circuit network*



Addressing

In a virtual-circuit network, two types of addressing are involved: global and local (virtual-circuit identifier).

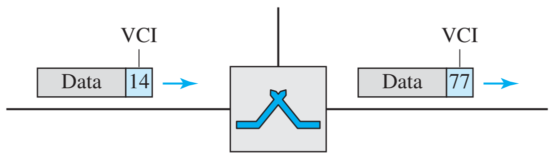
Global Addressing

A source or a destination needs to have a global address—an address that can be unique in the scope of the network or internationally if the network is part of an international network. However, we will see that a global address in virtual-circuit networks is used only to create a virtual-circuit identifier, as discussed next.

Virtual-Circuit Identifier

The identifier that is actually used for data transfer is called the ***virtual-circuit identifier* (VCI)** or the ***label***. A VCI, unlike a global address, is a small number that has only switch scope; it is used by a frame between two switches. When a frame arrives at a switch, it has a VCI; when it leaves, it has a different VCI. Figure 8.11 shows how the VCI in a data frame changes from one switch to another. Note that a VCI does not need to be a large number since each switch can use its own unique set of VCIs.

Figure 8.11 *Virtual-circuit identifier*



Three Phases

As in a circuit-switched network, a source and destination need to go through three phases in a virtual-circuit network: setup, data transfer, and teardown. In the setup phase, the source and destination use their global addresses to help switches make table entries for the connection. In the teardown phase, the source and destination inform the switches to delete the corresponding entry. Data transfer occurs between these two phases. We first discuss the data-transfer phase, which is more straightforward; we then talk about the setup and teardown phases.

Data-Transfer Phase

To transfer a frame from a source to its destination, all switches need to have a table entry for this virtual circuit. The table, in its simplest form, has four columns. This means that the switch holds four pieces of information for each virtual circuit that is already set up. We show later how the switches make their table entries, but for the moment we assume that each switch has a table with entries for all active virtual circuits. Figure 8.12 shows such a switch and its corresponding table.

Figure 8.12 *Switch and tables in a virtual-circuit network*

Incoming		Outgoing	
Port	VCI	Port	VCI
1	14	3	22
1	77	2	41

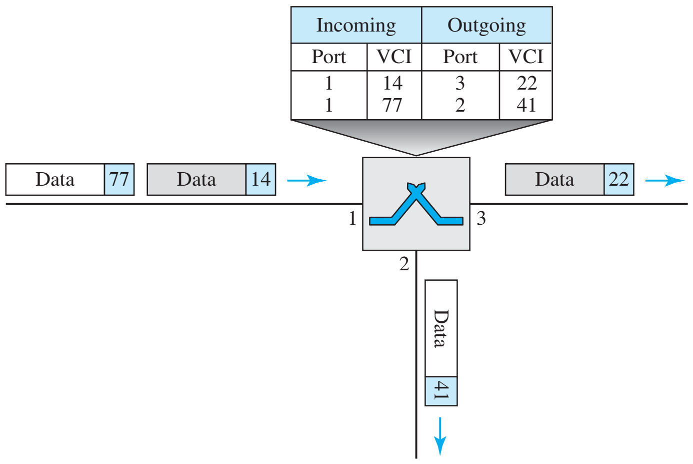


Figure 8.12 shows a frame arriving at port 1 with a VCI of 14. When the frame arrives, the switch looks in its table to find port 1 and a VCI of 14. When it is found, the switch knows to change the VCI to 22 and send out the frame from port 3.

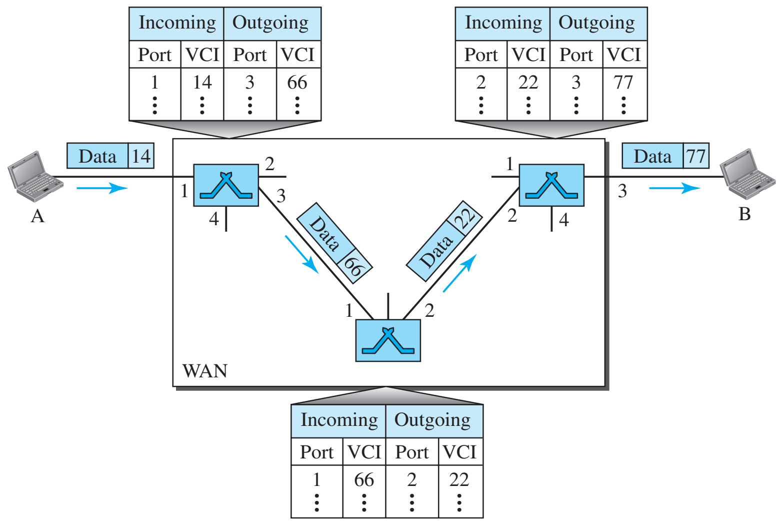
Figure 8.13 shows how a frame from source A reaches destination B and how its VCI changes during the trip. Each switch changes the VCI and routes the frame.

The data-transfer phase is active until the source sends all its frames to the destination. The procedure at the switch is the same for each frame of a message. The process creates a virtual circuit, not a real circuit, between the source and destination.

Setup Phase

In the setup phase, a switch creates an entry for a virtual circuit. For example, suppose source A needs to create a virtual circuit to B. Two steps are required: the setup request and the acknowledgment.

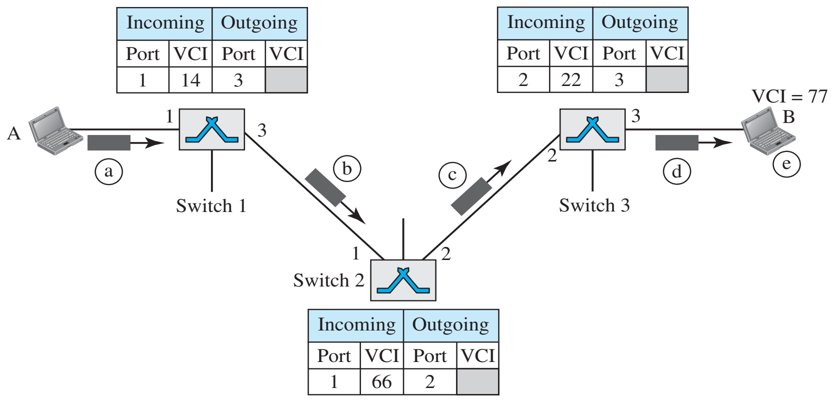
Figure 8.13 *Source-to-destination data transfer in a virtual-circuit network*



Setup Request

A setup request frame is sent from the source to the destination. Figure 8.14 shows the process.

Figure 8.14 *Setup request in a virtual-circuit network*



a. Source A sends a setup frame to switch 1.
b. Switch 1 receives the setup request frame. It knows that a frame going from A to B goes out through port 3. How the switch has obtained this information is a point covered in future chapters. The switch, in the setup phase, acts as a packet switch; it has a routing table which is different from the switching table. For the moment, assume that it knows the output port. The switch creates an entry in its table for this virtual circuit, but it is only able to fill three of the four columns. The switch assigns the incoming port (1) and chooses an available incoming VCI (14) and the

outgoing port (3). It does not yet know the outgoing VCI, which will be found during the acknowledgment step. The switch then forwards the frame through port 3 to switch 2.

c. Switch 2 receives the setup request frame. The same events happen here as at switch 1; three columns of the table are completed: in this case, incoming port (1), incoming VCI (66), and outgoing port (2).

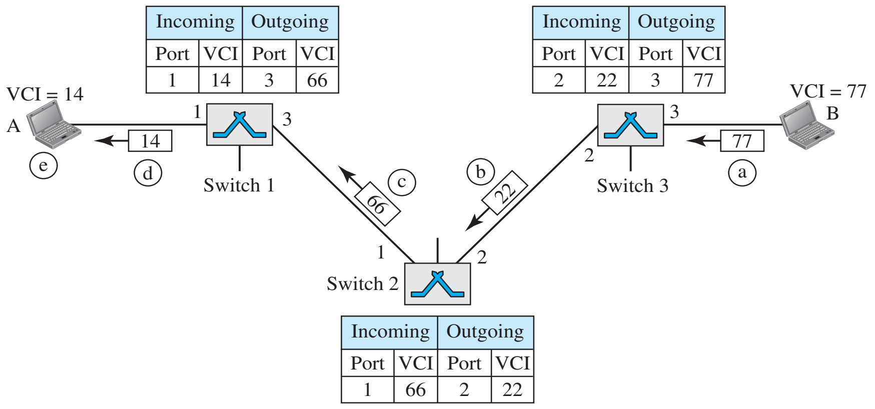
d. Switch 3 receives the setup request frame. Again, three columns are completed: incoming port (2), incoming VCI (22), and outgoing port (3).

e. Destination B receives the setup frame, and if it is ready to receive frames from A, it assigns a VCI to the incoming frames that come from A, in this case 77. This VCI lets the destination know that the frames come from A, and not other sources.

Acknowledgment

A special frame, called the *acknowledgment frame,* completes the entries in the switching tables. Figure 8.15 shows the process.

Figure 8.15 *Setup acknowledgment in a virtual-circuit network*



a. The destination sends an acknowledgment to switch 3. The acknowledgment carries the global source and destination addresses so the switch knows which entry in the table is to be completed. The frame also carries VCI 77, chosen by the destination as the incoming VCI for frames from A. Switch 3 uses this VCI to complete the outgoing VCI column for this entry. Note that 77 is the incoming VCI for destination B, but the outgoing VCI for switch 3.

b. Switch 3 sends an acknowledgment to switch 2 that contains its incoming VCI in the table, chosen in the previous step. Switch 2 uses this as the outgoing VCI in the table.

c. Switch 2 sends an acknowledgment to switch 1 that contains its incoming VCI in the table, chosen in the previous step. Switch 1 uses this as the outgoing VCI in the table.

d. Finally switch 1 sends an acknowledgment to source A that contains its incoming VCI in the table, chosen in the previous step.

e. The source uses this as the outgoing VCI for the data frames to be sent to destination B.

Teardown Phase

In this phase, source A, after sending all frames to B, sends a special frame called a *teardown request*. Destination B responds with a teardown confirmation frame. All switches delete the corresponding entry from their tables.

Efficiency

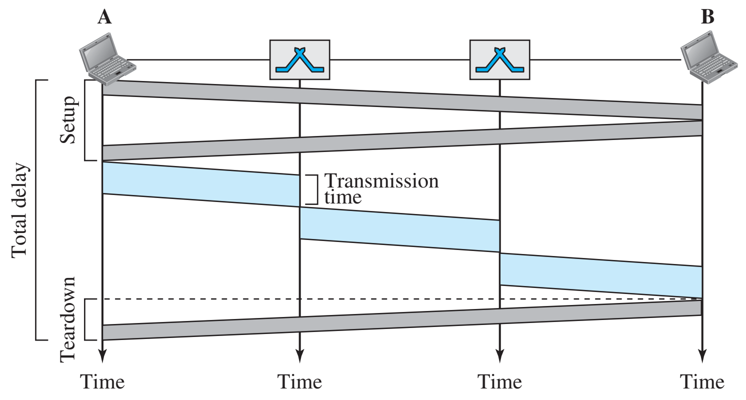
As we said before, resource reservation in a virtual-circuit network can be made during the setup or can be on demand during the data-transfer phase. In the first case, the delay for each packet is the same; in the second case, each packet may encounter different delays. There is one big advantage in a virtual-circuit network even if resource allocation is on demand. The source can check the availability of the resources, without actually reserving it. Consider a family that wants to dine at a restaurant. Although the restaurant may not accept reservations (allocation of the tables is on demand), the family can call and find out the waiting time. This can save the family time and effort.

In virtual-circuit switching, all packets belonging to the same source and destination travel the same path, but the packets may arrive at the destination with different delays if resource allocation is on demand.

Delay in Virtual-Circuit Networks

In a virtual-circuit network, there is a one-time delay for setup and a one-time delay for teardown. If resources are allocated during the setup phase, there is no wait time for individual packets. Figure 8.16 shows the delay for a packet traveling through two switches in a virtual-circuit network.

Figure 8.16 *Delay in a virtual-circuit network*



The packet is traveling through two switches (routers). There are three transmission times ($3T$), three propagation times (3τ), data transfer depicted by the sloping lines, a setup delay (which includes transmission and propagation in two directions), and a teardown delay (which includes transmission and propagation in one direction). We ignore the processing time in each switch. The total delay time is

Total delay + 3T + 3τ + setup delay + teardown delay

Circuit-Switched Technology in WANs

As we will see in Chapter 14, virtual-circuit networks are used in switched WANs such as ATM networks. The data-link layer of these technologies is well suited to the virtual-circuit technology.

Switching at the data-link layer in a switched WAN is normally implemented by using virtual-circuit techniques.

8.4 STRUCTURE OF A SWITCH

We use switches in circuit-switched and packet-switched networks. In this section, we discuss the structures of the switches used in each type of network.

8.4.1 Structure of Circuit Switches

Circuit switching today can use either of two technologies: the space-division switch or the time-division switch.

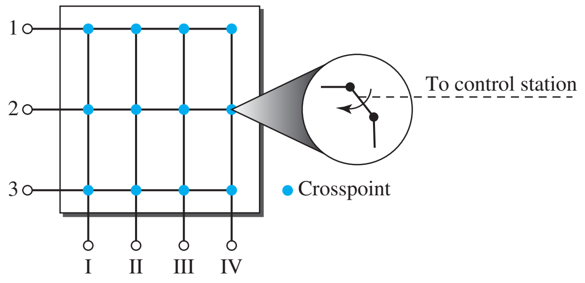
Space-Division Switch

In **space-division switching,** the paths in the circuit are separated from one another spatially. This technology was originally designed for use in analog networks but is used currently in both analog and digital networks. It has evolved through a long history of many designs.

Crossbar Switch

A **crossbar switch** connects n inputs to m outputs in a grid, using electronic micro-switches (transistors) at each **crosspoint** (see Figure 8.17). The major limitation of this design is the number of crosspoints required. To connect n inputs to m outputs using a

Figure 8.17 *Crossbar switch with three inputs and four outputs*

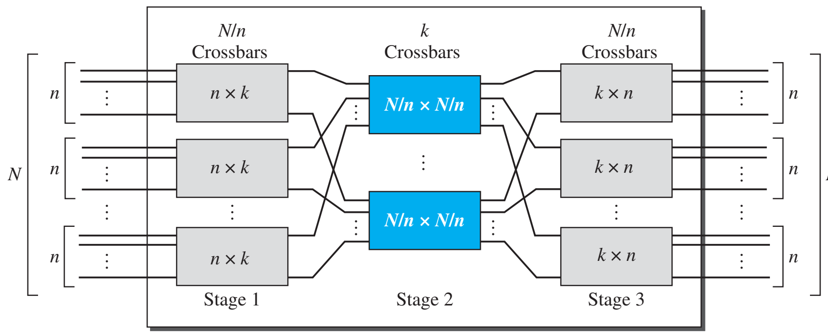


crossbar switch requires $n \times m$ crosspoints. For example, to connect 1000 inputs to 1000 outputs requires a switch with 1,000,000 crosspoints. A crossbar switch [?] with this number of crosspoints is impractical. Such a switch is also inefficient because statistics show that, in practice, fewer than 25 percent of the crosspoints are in use at any given time. The rest are idle.

Multistage Switch

The solution to the limitations of the crossbar switch is the **multistage switch,** which combines crossbar switches in several (normally three) stages, as shown in Figure 8.18. In a single crossbar switch, only one row or column (one path) is active for any connection. So we need $N \times N$ crosspoints. If we can allow multiple paths inside the switch, we can decrease the number of crosspoints. Each crosspoint in the middle stage can be accessed by multiple crosspoints in the first or third stage.

Figure 8.18 *Multistage switch*



To design a three-stage switch, we follow these steps:

1. We divide the N input lines into groups, each of n lines. For each group, we use one crossbar of size $n \times k$, where k is the number of crossbars in the middle stage. In other words, the first stage has N/n crossbars of $n \times k$ crosspoints.
2. We use k crossbars, each of size $(N/n) \times (N/n)$ in the middle stage.
3. We use N/n crossbars, each of size $k \times n$ at the third stage.

We can calculate the total number of crosspoints as follows:

$$\frac{N}{n}(n \times k) + k\left(\frac{N}{n} \times \frac{N}{n}\right) + \frac{N}{n}(k \times n) = 2kN + k\left(\frac{N}{n}\right)^2$$

In a three-stage switch, the total number of crosspoints is

$$2kN + k\left(\frac{N}{n}\right)^2$$

which is much smaller than the number of crosspoints in a single-stage switch (N^2).

Example 8.3

Design a three-stage, 200×200 switch ($N = 200$) with $k = 4$ and $n = 20$.

Solution

In the first stage we have N/n or 10 crossbars, each of size 20×4. In the second stage, we have 4 crossbars, each of size 10×10. In the third stage, we have 10 crossbars, each of size 4×20. The total number of crosspoints is $2kN + k(N/n)^2$, or 2000 crosspoints. This is 5 percent of the number of crosspoints in a single-stage switch ($200 \times 200 = 40{,}000$).

The multistage switch in Example 8.3 has one drawback—**blocking** during periods of heavy traffic. The whole idea of multistage switching is to share the crosspoints in the middle-stage crossbars. Sharing can cause a lack of availability if the resources are limited and all users want a connection at the same time. *Blocking* refers to times when one input cannot be connected to an output because there is no path available between them—all the possible intermediate switches are occupied.

In a single-stage switch, blocking does not occur because every combination of input and output has its own crosspoint; there is always a path. (Cases in which two inputs are trying to contact the same output do not count. That path is not blocked; the output is merely busy.) In the multistage switch described in Example 8.3, however, only four of the first 20 inputs can use the switch at a time, only four of the second 20 inputs can use the switch at a time, and so on. The small number of crossbars at the middle stage creates blocking.

In large systems, such as those having 10,000 inputs and outputs, the number of stages can be increased to cut down on the number of crosspoints required. As the number of stages increases, however, possible blocking increases as well. Many people have experienced blocking on public telephone systems in the wake of a natural disaster when the calls being made to check on or reassure relatives far outnumber the regular load of the system.

Clos investigated the condition of nonblocking in multistage switches and came up with the following formula. In a nonblocking switch, the number of middle-stage switches must be at least $2n - 1$. In other words, we need to have $k \geq 2n - 1$.

Note that the number of crosspoints is still smaller than that in a single-stage switch. Now we need to minimize the number of crosspoints with a fixed N by using the Clos criteria. We can take the derivative of the equation with respect to n (the only variable) and find the value of n that makes the result zero. This n must be equal to or greater than $(N/2)^{1/2}$. In this case, the total number of crosspoints is greater than or equal to $4N\,[(2N)^{1/2} - 1]$. In other words, the minimum number of crosspoints according to the Clos criteria is proportional to $N^{3/2}$.

According to Clos criterion: $n = (N/2)^{1/2}$ ***and*** $k \geq 2n - 1$
Total number of crosspoints $\geq 4N\,[(2N)^{1/2} - 1]$

Example 8.4

Redesign the previous three-stage, 200×200 switch, using the Clos criteria with a minimum number of crosspoints.

Solution

We let $n = (200/2)^{1/2}$, or $n = 10$. We calculate $k = 2n - 1 = 19$. In the first stage, we have 200/10, or 20, crossbars, each with 10×19 crosspoints. In the second stage, we have 19 crossbars, each with 10 × 10 crosspoints. In the third stage, we have 20 crossbars each with 19 × 10 crosspoints. The total number of crosspoints is 20(10 × 19) + 19(10 × 10) + 20(19 × 10) = 9500. If we use a single-stage switch, we need 200 × 200 = 40,000 crosspoints. The number of crosspoints in this three-stage switch is 24 percent that of a single-stage switch. More points are needed than in Example 8.3 (5 percent). The extra crosspoints are needed to prevent blocking.

A multistage switch that uses the Clos criteria and a minimum number of crosspoints still requires a huge number of crosspoints. For example, to have a 100,000 input/output switch, we need something close to 200 million crosspoints (instead of 10 billion). This means that if a telephone company needs to provide a switch to connect 100,000 telephones in a city, it needs 200 million crosspoints. The number can be reduced if we accept blocking. Today, telephone companies use time-division switching or a combination of space- and time-division switches, as we will see shortly.

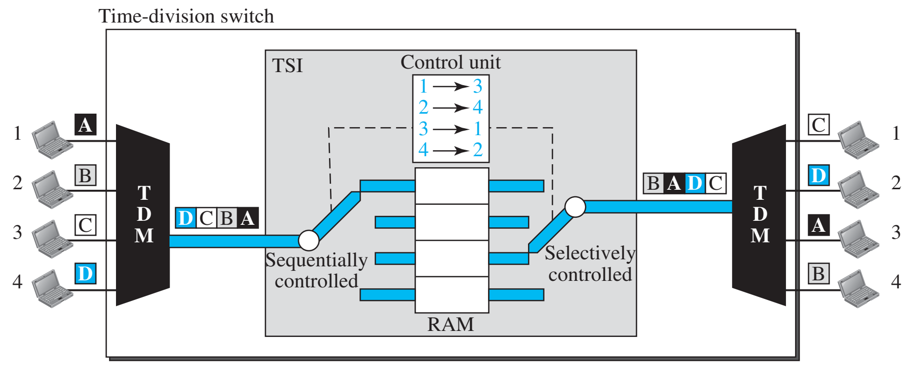
Time-Division Switch

Time-division switching uses time-division multiplexing (TDM) inside a switch. The most popular technology is called the **time-slot interchange (TSI).**

Time-Slot Interchange

Figure 8.19 shows a system connecting four input lines to four output lines. Imagine that each input line wants to send data to an output line according to the following pattern: (1 → 3), (2 → 4), (3 → 1), and (4 → 2), in which the arrow means "to."

Figure 8.19 *Time-slot interchange*



The figure combines a TDM multiplexer, a TDM demultiplexer, and a TSI consisting of random access memory (RAM) with several memory locations. The size of each location is the same as the size of a single time slot. The number of locations is the same as the number of inputs (in most cases, the numbers of inputs and outputs are equal). The RAM fills up with incoming data from time slots in the order received. Slots are then sent out in an order based on the decisions of a control unit.

Time- and Space-Division Switch Combinations

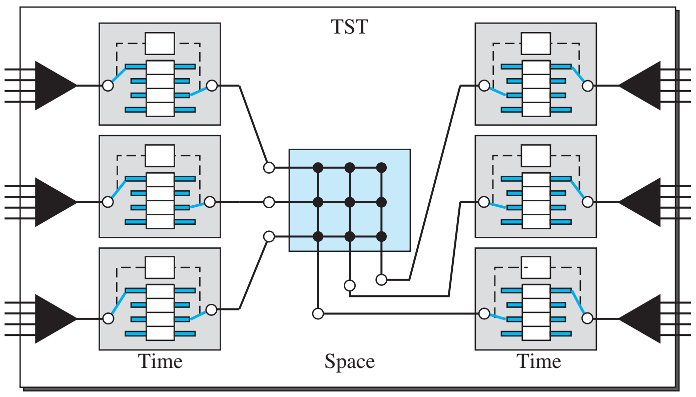
When we compare space-division and time-division switching, some interesting facts emerge. The advantage of space-division switching is that it is instantaneous. Its disadvantage is the number of crosspoints required to make space-division switching acceptable in terms of blocking.

The advantage of time-division switching is that it needs no crosspoints. Its disadvantage, in the case of TSI, is that processing each connection creates delays. Each time slot must be stored by the RAM, then retrieved and passed on.

In a third option, we combine space-division and time-division technologies to take advantage of the best of both. Combining the two results in switches that are optimized both physically (the number of crosspoints) and temporally (the amount of delay). Multistage switches of this sort can be designed as **time-space-time (TST) switches.**

Figure 8.20 shows a simple TST switch that consists of two time stages and one space stage and has 12 inputs and 12 outputs. Instead of one time-division switch, it divides the inputs into three groups (of four inputs each) and directs them to three time-slot interchanges. The result is that the average delay is one-third of what would result from using one time-slot interchange to handle all 12 inputs.

Figure 8.20 *Time-space-time switch*

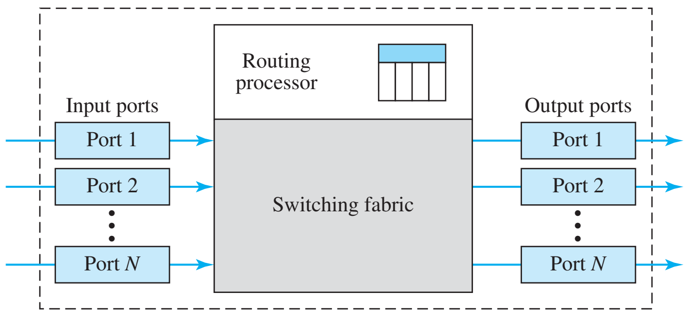


The last stage is a mirror image of the first stage. The middle stage is a space-division switch (crossbar) that connects the TSI groups to allow connectivity between all possible input and output pairs (e.g., to connect input 3 of the first group to output 7 of the second group).

8.4.2 Structure of Packet Switches

A switch used in a packet-switched network has a different structure from a switch used in a circuit-switched network.We can say that a packet switch has four components: **input ports, output ports,** the **routing processor,** and the **switching fabric,** as shown in Figure 8.21.

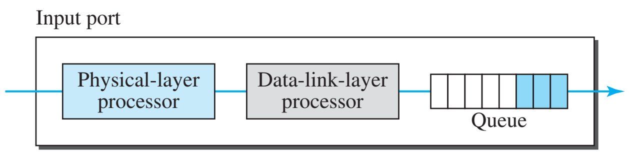
Figure 8.21 *Packet switch components*



Input Ports

An input port performs the physical and data-link functions of the packet switch. The bits are constructed from the received signal. The packet is decapsulated from the frame. Errors are detected and corrected. The packet is now ready to be routed by the network layer. In addition to a physical-layer processor and a data-link processor, the input port has buffers (queues) to hold the packet before it is directed to the switching fabric. Figure 8.22 shows a schematic diagram of an input port.

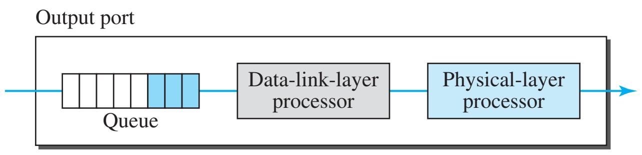
Figure 8.22 *Input port*



Output Port

The output port performs the same functions as the input port, but in the reverse order. First the outgoing packets are queued, then the packet is encapsulated in a frame, and finally the physical-layer functions are applied to the frame to create the signal to be sent on the line. Figure 8.23 shows a schematic diagram of an output port.

Figure 8.23 *Output port*



Routing Processor

The routing processor performs the functions of the network layer. The destination address is used to find the address of the next hop and, at the same time, the output port number from which the packet is sent out. This activity is sometimes referred to as **table lookup** because the routing processor searches the routing table. In the newer packet switches, this function of the routing processor is being moved to the input ports to facilitate and expedite the process.

Switching Fabrics

The most difficult task in a packet switch is to move the packet from the input queue to the output queue. The speed with which this is done affects the size of the input/output queue and the overall delay in packet delivery. In the past, when a packet switch was actually a dedicated computer, the memory of the computer or a bus was used as the switching fabric. The input port stored the packet in memory; the output port retrieved the packet from memory. Today, packet switches are specialized mechanisms that use a variety of switching fabrics. We briefly discuss some of these fabrics here.

Crossbar Switch

The simplest type of switching fabric is the crossbar switch, discussed in the previous section.

Banyan Switch

A more realistic approach than the crossbar switch is the **banyan switch** (named after the banyan tree). A banyan switch is a multistage switch with microswitches at each stage that route the packets based on the output port represented as a binary string. For n inputs and n outputs, we have $\log_2 n$ stages with $n/2$ microswitches at each stage. The first stage routes the packet based on the high-order bit of the binary string. The second stage routes the packet based on the second high-order bit, and so on. Figure 8.24 shows a banyan switch with eight inputs and eight outputs. The number of stages is $\log_2(8) = 3$.

Figure 8.24 *A banyan switch*

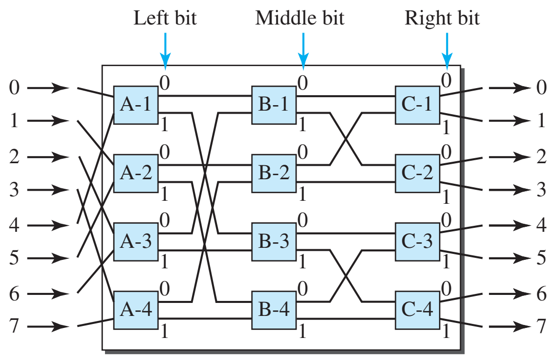
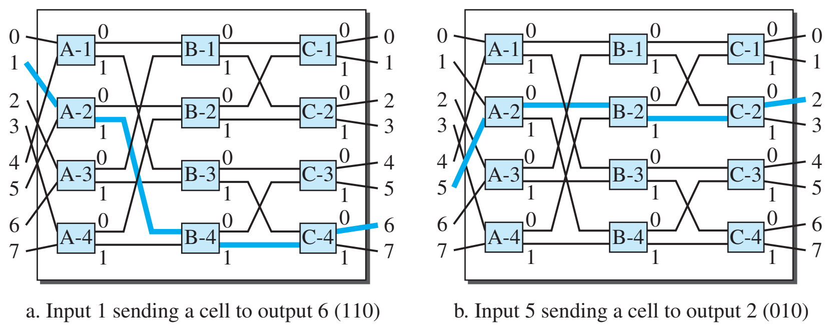


Figure 8.25 shows the operation. In part a, a packet has arrived at input port 1 and must go to output port 6 (110 in binary). The first microswitch (A-2) routes the packet

Figure 8.25 *Examples of routing in a banyan switch*

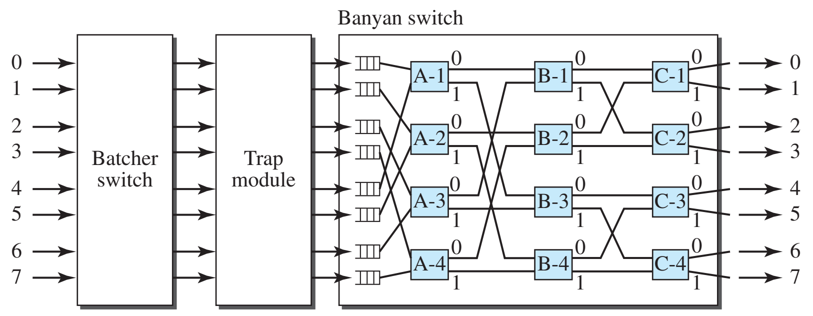

A-1
A-2
A-3
A-4
B-1
B-2
B-3
B-4
C-1
C-2
C-3
C-4
a. Input 1 sending a cell to output 6 (110)
b. Input 5 sending a cell to output 2 (010)

based on the first bit (1), the second microswitch (B-4) routes the packet based on the second bit (1), and the third microswitch (C-4) routes the packet based on the third bit (0). In part b, a packet has arrived at input port 5 and must go to output port 2 (010 in binary). The first microswitch (A-2) routes the packet based on the first bit (0), the second microswitch (B-2) routes the packet based on the second bit (1), and the third microswitch (C-2) routes the packet based on the third bit (0).

Batcher-Banyan Switch The problem with the banyan switch is the possibility of internal collision even when two packets are not heading for the same output port. We can solve this problem by sorting the arriving packets based on their destination port.

K. E. Batcher designed a switch that comes before the banyan switch and sorts the incoming packets according to their final destinations. The combination is called the **Batcher-banyan switch.** The sorting switch uses hardware merging techniques, but we do not discuss the details here. Normally, another hardware module called a **trap** is added between the Batcher switch and the banyan switch (see Figure 8.26) The trap module prevents duplicate packets (the packets with the same output destination) from passing to the banyan switch simultaneously. Only one packet for each destination is allowed at each tick; if there is more than one, they wait for the next tick.

Figure 8.26 *Batcher-banyan switch*



8.5 END-CHAPTER MATERIALS

8.5.1 Recommended Reading

For more details about subjects discussed in this chapter, we recommend the following books. The items in brackets [. . .] refer to the reference list at the end of the text.

Books

Switching is discussed in [Sta04] and [GW04]. Circuit-switching is fully discussed in [BEL01].

8.5.2 Key terms

banyan switch
Batcher-banyan switch
blocking
circuit switching
circuit-switched network
crossbar switch
crosspoint
data-transfer phase
datagram
datagram network
end system
input port
message switching
multistage switch
output port
packet switching
packet-switched network
routing processor
setup phase
space-division switching
switch
switching
switching fabric
table lookup
teardown phase
time-division switching
time-slot interchange (TSI)
time-space-time (TST) switch
trap
virtual-circuit identifier (VCI)
virtual-circuit network

8.5.3 Summary

A switched network consists of a series of interlinked nodes, called *switches*. Traditionally, three methods of switching have been important: circuit switching, packet switching, and message switching.

We can divide today's networks into three broad categories: circuit-switched networks, packet-switched networks, and message-switched networks. Packet-switched networks can also be divided into two subcategories: virtual-circuit networks and datagram networks. A circuit-switched network is made of a set of switches connected by physical links, in which each link is divided into *n* channels. Circuit switching takes place at the physical layer. In circuit switching, the resources need to be reserved during the setup phase; the resources remain dedicated for the entire duration of the data-transfer phase until the teardown phase.

In packet switching, there is no resource allocation for a packet. This means that there is no reserved bandwidth on the links, and there is no scheduled processing time for each packet. Resources are allocated on demand. In a datagram network, each packet is treated independently of all others. Packets in this approach are referred to as datagrams. There are no setup or teardown phases. A virtual-circuit network is a cross between a circuit-switched network and a datagram network. It has some characteristics of both. Circuit switching uses either of two technologies: the space-division switch or the time-division switch. A switch in a packet-switched network has a different structure from a switch used in a circuit-switched network. We can say that a packet switch has four types of components: input ports, output ports, a routing processor, and switching fabric.

8.6 PRACTICE SET

8.6.1 Quizzes

A set of interactive quizzes for this chapter can be found on the book website. It is strongly recommended that the student take the quizzes to check his/her understanding of the materials before continuing with the practice set.

8.6.2 Questions

Q8-1. Describe the need for switching and define a switch.

Q8-2. List the three traditional switching methods. Which are the most common today?

Q8-3. What are the two approaches to packet switching?

Q8-4. Compare and contrast a circuit-switched network and a packet-switched network.

Q8-5. What is the role of the address field in a packet traveling through a datagram network?

Q8-6. What is the role of the address field in a packet traveling through a virtual-circuit network?

Q8-7. Compare space-division and time-division switches.

Q8-8. What is TSI and what is its role in time-division switching?

Q8-9. Compare and contrast the two major categories of circuit switches.

Q8-10. List four major components of a packet switch and their functions.

8.6.3 Problems

P8-1. A path in a digital circuit-switched network has a data rate of 1 Mbps. The exchange of 1000 bits is required for the setup and teardown phases. The distance between two parties is 5000 km. Answer the following questions if the propagation speed is 2×10^8 m:

a. What is the total delay if 1000 bits of data are exchanged during the data-transfer phase?

b. What is the total delay if 100,000 bits of data are exchanged during the data-transfer phase?

c. What is the total delay if 1,000,000 bits of data are exchanged during the data-transfer phase?

d. Find the delay per 1000 bits of data for each of the above cases and compare them. What can you infer?

P8-2. Five equal-size datagrams belonging to the same message leave for the destination one after another. However, they travel through different paths as shown in Table 8.1.

Table 8.1 *P8-2*

Datagram	*Path Length*	*Visited Switches*
1	3200 km	1, 3, 5
2	11,700 km	1, 2, 5
3	12,200 km	1, 2, 3, 5
4	10,200 km	1, 4, 5
5	10,700 km	1, 4, 3, 5

We assume that the delay for each switch (including waiting and processing) is 3, 10, 20, 7, and 20 ms respectively. Assuming that the propagation speed is 2×10^8 m, find the order the datagrams arrive at the destination and the delay for each. Ignore any other delays in transmission.

P8-3. Transmission of information in any network involves end-to-end addressing and sometimes local addressing (such as VCI). Table 8.2 shows the types of networks and the addressing mechanism used in each of them.

Table 8.2 *P8-3*

Network	*Setup*	*Data Transfer*	*Teardown*
Circuit-switched	End-to-end	▨	End-to-end
Datagram	▨	End-to-end	▨
Virtual-circuit	End-to-end	Local	End-to-end

Answer the following questions:

a. Why does a circuit-switched network need end-to-end addressing during the setup and teardown phases? Why are no addresses needed during the data transfer phase for this type of network?

b. Why does a datagram network need only end-to-end addressing during the data transfer phase, but no addressing during the setup and teardown phases?

c. Why does a virtual-circuit network need addresses during all three phases?

P8-4. We mentioned that two types of networks, datagram and virtual-circuit, need a routing or switching table to find the output port from which the information belonging to a destination should be sent out, but a circuit-switched network has no need for such a table. Give the reason for this difference.

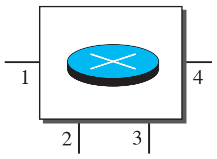
P8-5. An entry in the switching table of a virtual-circuit network is normally created during the setup phase and deleted during the teardown phase. In other words, the entries in this type of network reflect the current connections, the activity in the network. In contrast, the entries in a routing table of a datagram network do not depend on the current connections; they show the configuration of the network and how any packet should be routed to a final destination. The entries may remain the same even if there is no activity in the network. The routing tables, however, are updated if there are changes in the network. Can you explain the reason for these two different characteristics? Can we say that a virtual-circuit is a *connection-oriented* network and a datagram network is a *connectionless* network because of the above characteristics?

P8-6. The minimum number of columns in a datagram network is two; the minimum number of columns in a virtual-circuit network is four. Can you explain the reason? Is the difference related to the type of addresses carried in the packets of each network?

P8-7. Figure 8.27 shows a switch (router) in a datagram network.

Figure 8.27 *Problem P8-7*

Destination address	Output port
1233	3
1456	2
3255	1
4470	4
7176	2
8766	3
9144	2



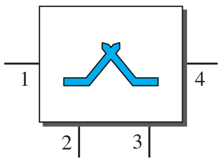
Find the output port for packets with the following destination addresses:

a. Packet 1: 7176
b. Packet 2: 1233
c. Packet 3: 8766
d. Packet 4: 9144

P8-8. Figure 8.28 shows a switch in a virtual-circuit network.

Figure 8.28 *Problem P8-8*

Incoming		Outgoing	
Port	VCI	Port	VCI
1	14	3	22
2	71	4	41
2	92	1	45
3	58	2	43
3	78	2	70
4	56	3	11



Find the output port and the output VCI for packets with the following input port and input VCI addresses:

a. Packet 1: 3, 78
b. Packet 2: 2, 92
c. Packet 3: 4, 56
d. Packet 4: 2, 71

P8-9. Answer the following questions:

a. Can a routing table in a datagram network have two entries with the same destination address? Explain.

b. Can a switching table in a virtual-circuit network have two entries with the same input port number? With the same output port number? With the same incoming VCIs? With the same outgoing VCIs? With the same incoming values (port, VCI)? With the same outgoing values (port, VCI)?

P8-10. It is obvious that a router or a switch needs to search to find information in the corresponding table. The searching in a routing table for a datagram network is based on the destination address; the searching in a switching table in a virtual-circuit network is based on the combination of incoming port and incoming VCI. Explain the reason and define how these tables must be ordered (sorted) based on these values.

P8-11. Consider an $n \times k$ crossbar switch with n inputs and k outputs.

a. Can we say that the switch acts as a multiplexer if $n > k$?

b. Can we say that the switch acts as a demultiplexer if $n < k$?

P8-12. We need a three-stage space-division switch with $N = 100$. We use 10 crossbars at the first and third stages and 4 crossbars at the middle stage.

a. Draw the configuration diagram.

b. Calculate the total number of crosspoints.

c. Find the possible number of simultaneous connections.

d. Find the possible number of simultaneous connections if we use a single crossbar (100×100).

e. Find the blocking factor, the ratio of the number of connections in part c and in part d.

P8-13. Repeat Problem 8-12 if we use 6 crossbars at the middle stage.

P8-14. Redesign the configuration of Problem 8-12 using the Clos criteria.

P8-15. We need to have a space-division switch with 1000 inputs and outputs. What is the total number of crosspoints in each of the following cases?

a. Using a single crossbar.

b. Using a multi-stage switch based on the Clos criteria.

P8-16. We need a three-stage time-space-time switch with $N = 100$. We use 10 TSIs at the first and third stages and 4 crossbars at the middle stage.

a. Draw the configuration diagram.

b. Calculate the total number of crosspoints.

c. Calculate the total number of memory locations we need for the TSIs.

8.7 SIMULATION EXPERIMENTS

8.7.1 Applets

We have created some Java applets to show some of the main concepts discussed in this chapter. It is strongly recommended that the students activate these applets on the book website and carefully examine the protocols in action.

CHAPTER 10

Error Detection and Correction

Networks must be able to transfer data from one device to another with acceptable accuracy. For most applications, a system must guarantee that the data received are identical to the data transmitted. Any time data are transmitted from one node to the next, they can become corrupted in passage. Many factors can alter one or more bits of a message. Some applications require a mechanism for detecting and correcting **errors.**

Some applications can tolerate a small level of error. For example, random errors in audio or video transmissions may be tolerable, but when we transfer text, we expect a very high level of accuracy.

At the data-link layer, if a frame is corrupted between the two nodes, it needs to be corrected before it continues its journey to other nodes. However, most link-layer protocols simply discard the frame and let the upper-layer protocols handle the retransmission of the frame. Some multimedia applications, however, try to correct the corrupted frame.

This chapter is divided into five sections.

❑ The first section introduces types of errors, the concept of redundancy, and distinguishes between error detection and correction.

❑ The second section discusses block coding. It shows how error can be detected using block coding and also introduces the concept of Hamming distance.

❑ The third section discusses cyclic codes. It discusses a subset of cyclic code, CRC, that is very common in the data-link layer. The section shows how CRC can be easily implemented in hardware and represented by polynomials.

❑ The fourth section discusses checksums. It shows how a checksum is calculated for a set of data words. It also gives some other approaches to traditional checksum.

❑ The fifth section discusses forward error correction. It shows how Hamming distance can also be used for this purpose. The section also describes cheaper methods to achieve the same goal, such as XORing of packets, interleaving chunks, or compounding high and low resolutions packets.

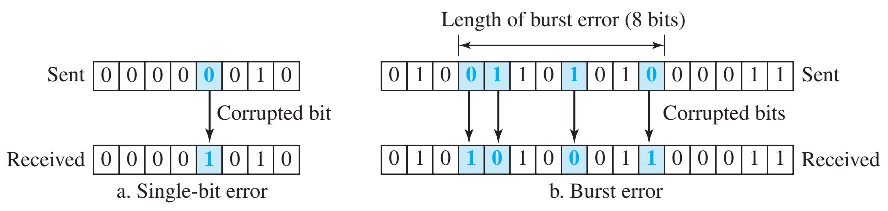
10.1 INTRODUCTION

Let us first discuss some issues related, directly or indirectly, to error detection and correction.

10.1.1 Types of Errors

Whenever bits flow from one point to another, they are subject to unpredictable changes because of **interference.** This interference can change the shape of the signal. The term ***single-bit error*** means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1. The term ***burst error*** means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1. Figure 10.1 shows the effect of a single-bit and a burst error on a data unit.

Figure 10.1 *Single-bit and burst error*



A burst error is more likely to occur than a single-bit error because the duration of the noise signal is normally longer than the duration of 1 bit, which means that when noise affects data, it affects a set of bits. The number of bits affected depends on the data rate and duration of noise. For example, if we are sending data at 1 kbps, a noise of 1/100 second can affect 10 bits; if we are sending data at 1 Mbps, the same noise can affect 10,000 bits.

10.1.2 Redundancy

The central concept in detecting or correcting errors is **redundancy**. To be able to detect or correct errors, we need to send some extra bits with our data. These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct corrupted bits.

10.1.3 Detection versus Correction

The correction of errors is more difficult than the detection. In **error detection**, we are only looking to see if any error has occurred. The answer is a simple yes or no. We are not even interested in the number of corrupted bits. A single-bit error is the same for us as a burst error. In **error correction**, we need to know the exact number of bits that are corrupted and, more importantly, their location in the message. The number of errors and the size of the message are important factors. If we need to correct a single error in an 8-bit data unit, we need to consider eight possible error locations; if we need to correct two errors in a data unit of the same size, we need to consider 28 (permutation of 8 by 2) possibilities. You can imagine the receiver's difficulty in finding 10 errors in a data unit of 1000 bits.

10.1.4 Coding

Redundancy is achieved through various coding schemes. The sender adds redundant bits through a process that creates a relationship between the redundant bits and the actual data bits. The receiver checks the relationships between the two sets of bits to detect errors. The ratio of redundant bits to data bits and the robustness of the process are important factors in any coding scheme.

We can divide coding schemes into two broad categories: **block coding** and **convolution coding**. In this book, we concentrate on block coding; convolution coding is more complex and beyond the scope of this book.

10.2 BLOCK CODING

In block coding, we divide our message into blocks, each of k bits, called ***datawords.*** We add r redundant bits to each block to make the length $n = k + r$. The resulting n-bit blocks are called ***codewords.*** How the extra r bits are chosen or calculated is something we will discuss later. For the moment, it is important to know that we have a set of datawords, each of size k, and a set of codewords, each of size of n. With k bits, we can create a combination of 2^k datawords; with n bits, we can create a combination of 2^n codewords. Since $n > k$, the number of possible codewords is larger than the number of possible datawords. The block coding process is one-to-one; the same dataword is always encoded as the same codeword. This means that we have $2^n - 2^k$ codewords that are not used. We call these codewords invalid or illegal. The trick in error detection is the existence of these invalid codes, as we discuss next. If the receiver receives an invalid codeword, this indicates that the data was corrupted during transmission.

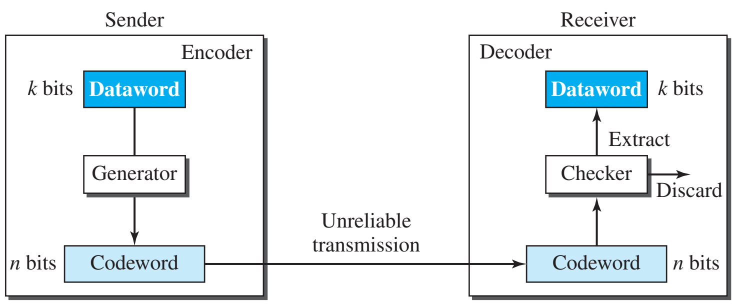
10.2.1 Error Detection

How can errors be detected by using block coding? If the following two conditions are met, the receiver can detect a change in the original codeword.

1. The receiver has (or can find) a list of valid codewords.
2. The original codeword has changed to an invalid one.

Figure 10.2 shows the role of block coding in error detection. The sender creates codewords out of datawords by using a generator that applies the rules and procedures of encoding (discussed later). Each codeword sent to the receiver may change during transmission. If the received codeword is the same as one of the valid codewords, the word is accepted; the corresponding dataword is extracted for use. If the received codeword is not valid, it is discarded. However, if the codeword is corrupted during transmission but the received word still matches a valid codeword, the error remains undetected.

Figure 10.2 *Process of error detection in block coding*



Example 10.1

Let us assume that $k = 2$ and $n = 3$. Table 10.1 shows the list of datawords and codewords. Later, we will see how to derive a codeword from a dataword.

Table 10.1 *A code for error detection in Example 10.1*

Dataword	*Codeword*	*Dataword*	*Codeword*
00	000	10	101
01	011	11	110

Assume the sender encodes the dataword 01 as 011 and sends it to the receiver. Consider the following cases:

1. The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.
2. The codeword is corrupted during transmission, and 111 is received (the leftmost bit is corrupted). This is not a valid codeword and is discarded.
3. The codeword is corrupted during transmission, and 000 is received (the right two bits are corrupted). This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.

An error-detecting code can detect only the types of errors for which it is designed; other types of errors may remain undetected.

Hamming Distance

One of the central concepts in coding for error control is the idea of the Hamming distance. The **Hamming distance** between two words (of the same size) is the number of differences between the corresponding bits. We show the Hamming distance between two words x and y as $\boldsymbol{d}(x, y)$. We may wonder why Hamming distance is important for error detection. The reason is that the Hamming distance between the received codeword and the sent codeword is the number of bits that are corrupted during transmission. For example, if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Hamming distance between the two is $\boldsymbol{d}(00000, 01101) = 3$. In other words, if the Hamming distance between the sent and the received codeword is not zero, the codeword has been corrupted during transmission.

The Hamming distance can easily be found if we apply the XOR operation (⊕) on the two words and count the number of 1s in the result. Note that the Hamming distance is a value greater than or equal to zero.

The Hamming distance between two words is the number of differences between corresponding bits.

Example 10.2

Let us find the Hamming distance between two pairs of words.

1. The Hamming distance ***d***(000, 011) is 2 because (000 ⊕ 011) is 011 (two 1s).
2. The Hamming distance ***d***(10101, 11110) is 3 because (10101 ⊕ 11110) is 01011 (three 1s).

Minimum Hamming Distance for Error Detection

In a set of codewords, the **minimum Hamming distance** is the smallest Hamming distance between all possible pairs of codewords. Now let us find the minimum Hamming distance in a code if we want to be able to detect up to s errors. If s errors occur during transmission, the Hamming distance between the sent codeword and received codeword is s. If our system is to detect up to s errors, the minimum distance between the valid codes must be $(s + 1)$, so that the received codeword does not match a valid codeword. In other words, if the minimum distance between all valid codewords is $(s + 1)$, the received codeword cannot be erroneously mistaken for another codeword. The error will be detected. We need to clarify a point here: Although a code with $d_{\min} = s + 1$ may be able to detect more than s errors in some special cases, only s or fewer errors are guaranteed to be detected.

To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{\min} = s + 1$.

We can look at this criteria geometrically. Let us assume that the sent codeword x is at the center of a circle with radius s. All received codewords that are created by 0 to s errors are points inside the circle or on the perimeter of the circle. All other valid codewords must be outside the circle, as shown in Figure 10.3. This means that $d_{\min}$ must be an integer greater than s or $d_{\min} = s + 1$.

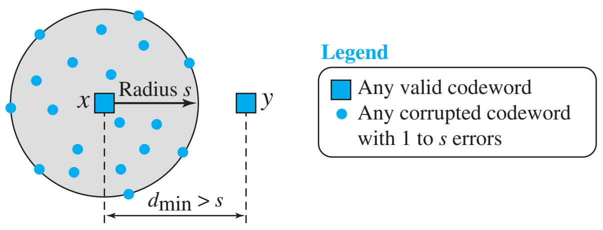
Example 10.3

The minimum Hamming distance for our first code scheme (Table 10.1) is 2. This code guarantees detection of only a single error. For example, if the third codeword (101) is sent and one error occurs, the received codeword does not match any valid codeword. If two errors occur, however, the received codeword may match a valid codeword and the errors are not detected.

Example 10.4

A code scheme has a Hamming distance $d_{\min} = 4$. This code guarantees the detection of up to three errors ($d = s + 1$ or $s = 3$).

Figure 10.3 *Geometric concept explaining* d_{min} *in error detection*



Linear Block Codes

Almost all block codes used today belong to a subset of block codes called ***linear block codes***. The use of nonlinear block codes for error detection and correction is not as widespread because their structure makes theoretical analysis and implementation difficult. We therefore concentrate on linear block codes. The formal definition of linear block codes requires the knowledge of abstract algebra (particularly Galois fields), which is beyond the scope of this book. We therefore give an informal definition. For our purposes, a linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.

Example 10.5

The code in Table 10.1 is a linear block code because the result of XORing any codeword with any other codeword is a valid codeword. For example, the XORing of the second and third codewords creates the fourth one.

Minimum Distance for Linear Block Codes

It is simple to find the minimum Hamming distance for a linear block code. The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.

Example 10.6

In our first code (Table 10.1), the numbers of 1s in the nonzero codewords are 2, 2, and 2. So the minimum Hamming distance is $d_{\min} = 2$.

Parity-Check Code

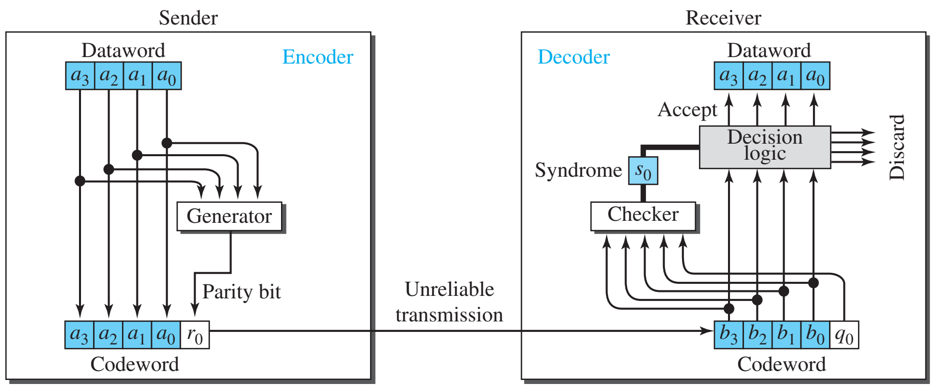
Perhaps the most familiar error-detecting code is the **parity-check code.** This code is a linear block code. In this code, a k-bit dataword is changed to an n-bit codeword where $n = k + 1$. The extra bit, called the *parity bit,* is selected to make the total number of 1s in the codeword even. Although some implementations specify an odd number of 1s, we discuss the even case. The minimum Hamming distance for this category is $d_{\min} = 2$, which means that the code is a single-bit error-detecting code. Our first code (Table 10.1) is a parity-check code ($k = 2$ and $n = 3$). The code in Table 10.2 is also a parity-check code with $k = 4$ and $n = 5$.

Table 10.2 *Simple parity-check code C(5, 4)*

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	**00000**	1000	**10001**
0001	**00011**	1001	**10010**
0010	**00101**	1010	**10100**
0011	**00110**	1011	**10111**
0100	**01001**	1100	**11000**
0101	**01010**	1101	**11011**
0110	**01100**	1110	**11101**
0111	**01111**	1111	**11110**

Figure 10.4 shows a possible structure of an encoder (at the sender) and a decoder (at the receiver).

Figure 10.4 *Encoder and decoder for simple parity-check code*



The calculation is done in **modular arithmetic** (see Appendix E). The encoder uses a generator that takes a copy of a 4-bit dataword (a_0, a_1, a_2, and a_3) and generates a parity bit r_0. The dataword bits and the parity bit create the 5-bit codeword. The parity bit that is added makes the number of 1s in the codeword even. This is normally done by adding the 4 bits of the dataword (modulo-2); the result is the parity bit. In other words,

$$r_0 = a_3 + a_2 + a_1 + a_0 \qquad \textbf{(modulo-2)}$$

If the number of 1s is even, the result is 0; if the number of 1s is odd, the result is 1. In both cases, the total number of 1s in the codeword is even.

The sender sends the codeword, which may be corrupted during transmission. The receiver receives a 5-bit word. The checker at the receiver does the same thing as the generator in the sender with one exception: The addition is done over all 5 bits. The result, which is called the ***syndrome,*** is just 1 bit. The syndrome is 0 when the number of 1s in the received codeword is even; otherwise, it is 1.

$$s_0 = b_3 + b_2 + b_1 + b_0 + q_0 \qquad \textbf{(modulo-2)}$$

The syndrome is passed to the decision logic analyzer. If the syndrome is 0, there is no detectable error in the received codeword; the data portion of the received codeword is accepted as the dataword; if the syndrome is 1, the data portion of the received codeword is discarded. The dataword is not created.

Example 10.7

Let us look at some transmission scenarios. Assume the sender sends the dataword 1011. The codeword created from this dataword is 10111, which is sent to the receiver. We examine five cases:

1. No error occurs; the received codeword is 10111. The syndrome is 0. The dataword 1011 is created.
2. One single-bit error changes a_1. The received codeword is 10**0**11. The syndrome is 1. No dataword is created.
3. One single-bit error changes r_0. The received codeword is 1011**0**. The syndrome is 1. No dataword is created. Note that although none of the dataword bits are corrupted, no dataword is created because the code is not sophisticated enough to show the position of the corrupted bit.
4. An error changes r_0 and a second error changes a_3. The received codeword is **0**011**0**. The syndrome is 0. The dataword 0011 is created at the receiver. Note that here the dataword is wrongly created due to the syndrome value. The simple parity-check decoder cannot detect an even number of errors. The errors cancel each other out and give the syndrome a value of 0.
5. Three bits—a_3, a_2, and a_1—are changed by errors. The received codeword is **010**11. The syndrome is 1. The dataword is not created. This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

A parity-check code can detect an odd number of errors.

10.3 CYCLIC CODES

Cyclic codes are special linear block codes with one extra property. In a **cyclic code,** if a codeword is cyclically shifted (rotated), the result is another codeword. For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword. In this case, if we call the bits in the first word a_0 to a_6, and the bits in the second word b_0 to b_6, we can shift the bits by using the following:

$$b_1 = a_0 \qquad b_2 = a_1 \qquad b_3 = a_2 \qquad b_4 = a_3 \qquad b_5 = a_4 \qquad b_6 = a_5 \qquad b_0 = a_6$$

In the rightmost equation, the last bit of the first word is wrapped around and becomes the first bit of the second word.

10.3.1 Cyclic Redundancy Check

We can create cyclic codes to correct errors. However, the theoretical background required is beyond the scope of this book. In this section, we simply discuss a subset of cyclic codes called the **cyclic redundancy check (CRC)**, which is used in networks such as LANs and WANs.

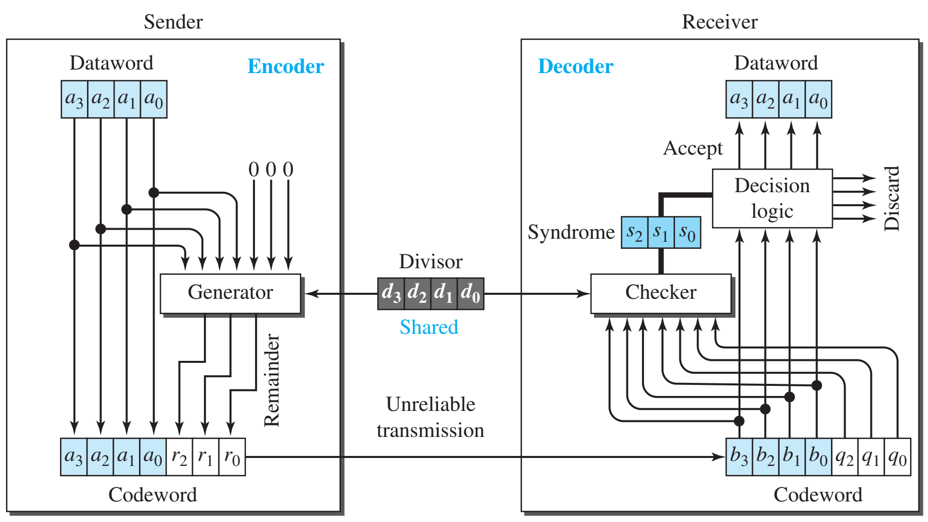
Table 10.3 shows an example of a CRC code. We can see both the linear and cyclic properties of this code.

Table 10.3 *A CRC code with C(7, 4)*

Dataword	*Codeword*	*Dataword*	*Codeword*
0000	0000000	1000	1000101
0001	0001011	1001	1001110
0010	0010110	1010	1010011
0011	0011101	1011	1011000
0100	0100111	1100	1100010
0101	0101100	1101	1101001
0110	0110001	1110	1110100
0111	0111010	1111	1111111

Figure 10.5 shows one possible design for the encoder and decoder.

Figure 10.5 *CRC encoder and decoder*



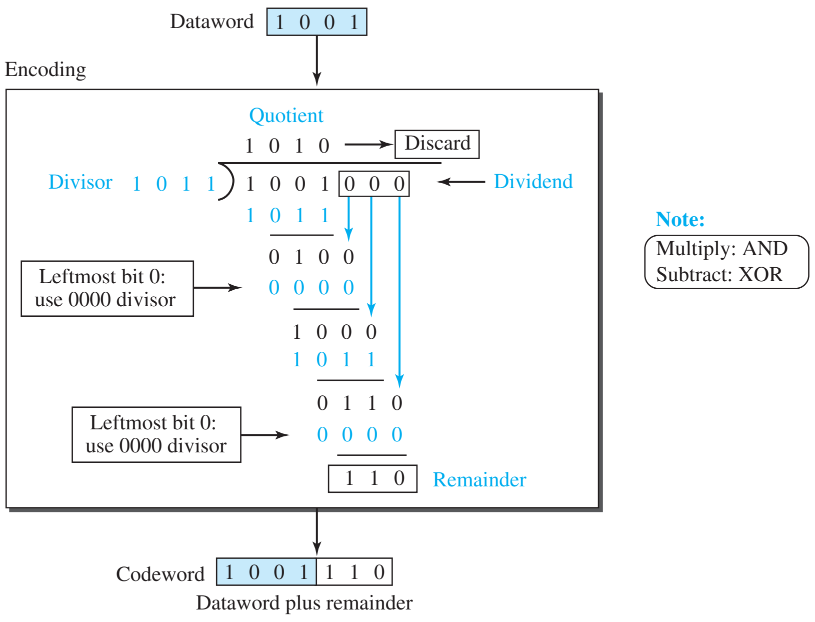
In the encoder, the dataword has k bits (4 here); the codeword has n bits (7 here). The size of the dataword is augmented by adding $n - k$ (3 here) 0s to the right-hand side of the word. The n-bit result is fed into the generator. The generator uses a divisor of size $n - k + 1$ (4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division). The quotient of the division is discarded; the remainder ($r_2r_1r_0$) is appended to the dataword to create the codeword.

The decoder receives the codeword (possibly corrupted in transition). A copy of all n bits is fed to the checker, which is a replica of the generator. The remainder produced by the checker is a syndrome of $n - k$ (3 here) bits, which is fed to the decision logic analyzer. The analyzer has a simple function. If the syndrome bits are all 0s, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded (error).

Encoder

Let us take a closer look at the encoder. The encoder takes a dataword and augments it with $n - k$ number of 0s. It then divides the augmented dataword by the divisor, as shown in Figure 10.6.

Figure 10.6 *Division in CRC encoder*



The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. However, addition and subtraction in this case are the same; we use the XOR operation to do both.

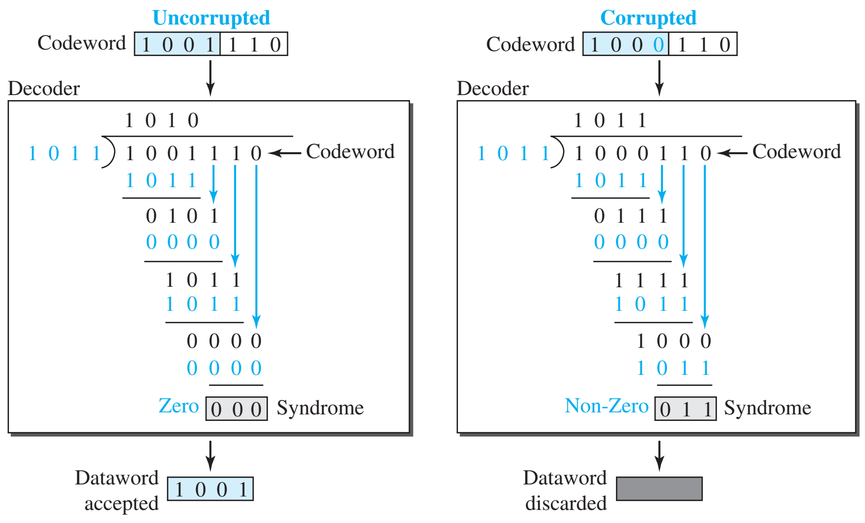
As in decimal division, the process is done step by step. In each step, a copy of the divisor is XORed with the 4 bits of the dividend. The result of the XOR operation (remainder) is 3 bits (in this case), which is used for the next step after 1 extra bit is pulled down to make it 4 bits long. There is one important point we need to remember in this type of division. If the leftmost bit of the dividend (or the part used in each step) is 0, the step cannot use the regular divisor; we need to use an all-0s divisor.

When there are no bits left to pull down, we have a result. The 3-bit remainder forms the **check bits** (r_2, r_1, and r_0). They are appended to the dataword to create the codeword.

Decoder

The codeword can change during transmission. The decoder does the same division process as the encoder. The remainder of the division is the syndrome. If the syndrome is all 0s, there is no error with a high probability; the dataword is separated from the received codeword and accepted. Otherwise, everything is discarded. Figure 10.7 shows two cases: The left-hand figure shows the value of the syndrome when no error has occurred; the syndrome is 000. The right-hand part of the figure shows the case in which there is a single error. The syndrome is not all 0s (it is 011).

Figure 10.7 *Division in the CRC decoder for two cases*



Divisor

We may be wondering how the divisor 1011 is chosen. This depends on the expectation we have from the code. We will show some standard divisors later in the chapter (Table 10.4) after we discuss polynomials.

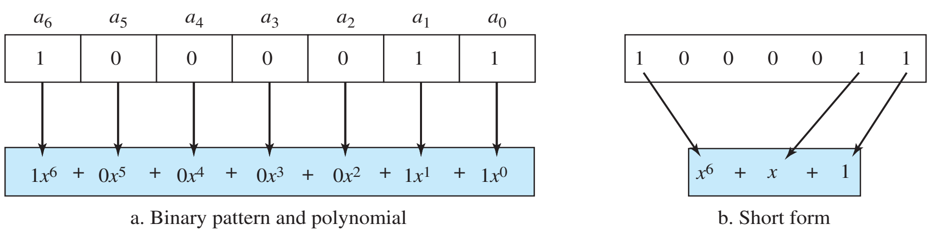
10.3.2 Polynomials

A better way to understand cyclic codes and how they can be analyzed is to represent them as polynomials. Again, this section is optional.

A pattern of 0s and 1s can be represented as a **polynomial** with coefficients of 0 and 1. The power of each term shows the position of the bit; the coefficient shows the value of the bit. Figure 10.8 shows a binary pattern and its polynomial representation. In Figure 10.8a we show how to translate a binary pattern into a polynomial; in Figure 10.8b we show how the polynomial can be shortened by removing all terms with zero coefficients and replacing x^1 by x and x^0 by 1.

Figure 10.8 shows one immediate benefit; a 7-bit pattern can be replaced by three terms. The benefit is even more conspicuous when we have a polynomial such as

Figure 10.8 *A polynomial to represent a binary word*


a_6 a_5 a_4 a_3 a_2 a_1 a_0
1 0 0 0 0 1 1
$1x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0$
a. Binary pattern and polynomial
1 0 0 0 0 1 1
$x^6 + x + 1$
b. Short form

$x^{23} + x^3 + 1$. Here the bit pattern is 24 bits in length (three 1s and twenty-one 0s) while the polynomial is just three terms.

Degree of a Polynomial

The degree of a polynomial is the highest power in the polynomial. For example, the degree of the polynomial $x^6 + x + 1$ is 6. Note that the degree of a polynomial is 1 less than the number of bits in the pattern. The bit pattern in this case has 7 bits.

Adding and Subtracting Polynomials

Adding and subtracting polynomials in mathematics are done by adding or subtracting the coefficients of terms with the same power. In our case, the coefficients are only 0 and 1, and adding is in modulo-2. This has two consequences. First, addition and subtraction are the same. Second, adding or subtracting is done by combining terms and deleting pairs of identical terms. For example, adding $x^5 + x^4 + x^2$ and $x^6 + x^4 + x^2$ gives just $x^6 + x^5$. The terms x^4 and x^2 are deleted. However, note that if we add, for example, three polynomials and we get x^2 three times, we delete a pair of them and keep the third.

Multiplying or Dividing Terms

In this arithmetic, multiplying a term by another term is very simple; we just add the powers. For example, $x^3 \times x^4$ is x^7. For dividing, we just subtract the power of the second term from the power of the first. For example, x^5/x^2 is x^3.

Multiplying Two Polynomials

Multiplying a polynomial by another is done term by term. Each term of the first polynomial must be multiplied by all terms of the second. The result, of course, is then simplified, and pairs of equal terms are deleted. The following is an example:

$$\begin{aligned}(x^5 + x^3 + x^2 + x)(x^2 + x + 1) &= x^7 + x^6 + x^5 + x^5 + x^4 + x^3 + x^4 + x^3 + x^2 + x^3 + x^2 + x \\ &= x^7 + x^6 + x^3 + x\end{aligned}$$

Dividing One Polynomial by Another

Division of polynomials is conceptually the same as the binary division we discussed for an encoder. We divide the first term of the dividend by the first term of the divisor to get the first term of the quotient. We multiply the term in the quotient by the divisor and subtract the result from the dividend. We repeat the process until the dividend degree is less than the divisor degree. We will show an example of division later in this chapter.

Shifting

A binary pattern is often shifted a number of bits to the right or left. Shifting to the left means adding extra 0s as rightmost bits; shifting to the right means deleting some rightmost bits. Shifting to the left is accomplished by multiplying each term of the polynomial by x^m, where m is the number of shifted bits; shifting to the right is accomplished by dividing each term of the polynomial by x^m. The following shows shifting to the left and to the right. Note that we do not have negative powers in the polynomial representation.

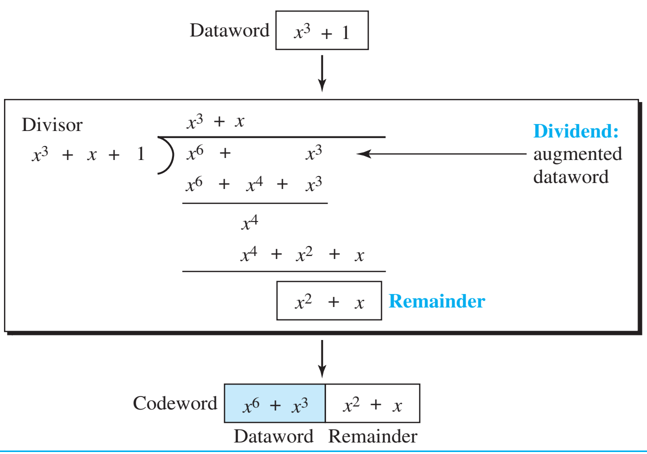
Shifting left 3 bits: 10011 becomes 10011000	$x^4 + x + 1$ becomes $x^7 + x^4 + x^3$
Shifting right 3 bits: 10011 becomes 10	$x^4 + x + 1$ becomes x

When we augmented the dataword in the encoder of Figure 10.6, we actually shifted the bits to the left. Also note that when we concatenate two bit patterns, we shift the first polynomial to the left and then add the second polynomial.

10.3.3 Cyclic Code Encoder Using Polynomials

Now that we have discussed operations on polynomials, we show the creation of a codeword from a dataword. Figure 10.9 is the polynomial version of Figure 10.6. We can see that the process is shorter. The dataword 1001 is represented as $x^3 + 1$. The divisor 1011 is represented as $x^3 + x + 1$. To find the augmented dataword, we have left-shifted the dataword 3 bits (multiplying by x^3). The result is $x^6 + x^3$. Division is straightforward. We divide the first term of the dividend, x^6, by the first term of the divisor, x^3. The first term of the quotient is then x^6/x^3, or x^3. Then we multiply x^3 by the divisor and subtract (according to our previous definition of subtraction) the result from the dividend. The result is x^4, with a degree greater than the divisor's degree; we continue to divide until the degree of the remainder is less than the degree of the divisor.

Figure 10.9 *CRC division using polynomials*



It can be seen that the polynomial representation can easily simplify the operation of division in this case, because the two steps involving all-0s divisors are not needed here. (Of course, one could argue that the all-0s divisor step can also be eliminated in binary division.) In a polynomial representation, the divisor is normally referred to as the ***generator polynomial*** $t(x)$.

The divisor in a cyclic code is normally called the *generator polynomial* or simply the *generator.*

10.3.4 Cyclic Code Analysis

We can analyze a cyclic code to find its capabilities by using polynomials. We define the following, where $f(x)$ is a polynomial with binary coefficients.

Dataword: $d(x)$ **Codeword:** $c(x)$ **Generator:** $g(x)$ **Syndrome:** $s(x)$ **Error:** $e(x)$

If $s(x)$ is not zero, then one or more bits is corrupted. However, if $s(x)$ is zero, either no bit is corrupted or the decoder failed to detect any errors. (Note that ¦ means divide).

In a cyclic code,

1. If $s(x)$ ¦ 0, one or more bits is corrupted.
2. If $s(x) = 0$, either
   a. No bit is corrupted, or
   b. Some bits are corrupted, but the decoder failed to detect them.

In our analysis we want to find the criteria that must be imposed on the generator, $g(x)$ to detect the type of error we especially want to be detected. Let us first find the relationship among the sent codeword, error, received codeword, and the generator. We can say

$$\textbf{Received codeword} = c(x) + e(x)$$

In other words, the received codeword is the sum of the sent codeword and the error. The receiver divides the received codeword by $g(x)$ to get the syndrome. We can write this as

$$\frac{\textbf{Received codeword}}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$$

The first term at the right-hand side of the equality has a remainder of zero (according to the definition of codeword). So the syndrome is actually the remainder of the second term on the right-hand side. If this term does not have a remainder (syndrome = 0), either $e(x)$ is 0 or $e(x)$ is divisible by $g(x)$. We do not have to worry about the first case (there is no error); the second case is very important. Those errors that are divisible by $g(x)$ are not caught.

In a cyclic code, those $e(x)$ errors that are divisible by $g(x)$ are not caught.

Let us show some specific errors and see how they can be caught by a well-designed $g(x)$.

Single-Bit Error

What should the structure of $g(x)$ be to guarantee the detection of a single-bit error? A single-bit error is $e(x) = x^i$, where i is the position of the bit. If a single-bit error is caught, then x^i is not divisible by $g(x)$. (Note that when we say *not divisible,* we mean that there is a remainder.) If $g(x)$ has at least two terms (which is normally the case) and the coefficient of x^0 is not zero (the rightmost bit is 1), then $e(x)$ cannot be divided by $g(x)$.

If the generator has more than one term and the coefficient of x^0 is 1, all single-bit errors can be caught.

Example 10.8

Which of the following $g(x)$ values guarantees that a single-bit error is caught? For each case, what is the error that cannot be caught?

a. $x + 1$
b. x^3
c. 1

Solution

a. No x^i can be divisible by $x + 1$. In other words, $x^i/(x + 1)$ always has a remainder. So the syndrome is nonzero. Any single-bit error can be caught.

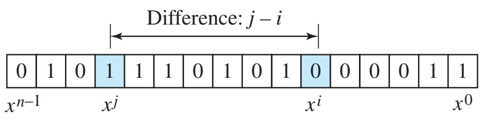
b. If i is equal to or greater than 3, x^i is divisible by $g(x)$. The remainder of x^i/x^3 is zero, and the receiver is fooled into believing that there is no error, although there might be one. Note that in this case, the corrupted bit must be in position 4 or above. All single-bit errors in positions 1 to 3 are caught.

c. All values of i make x^i divisible by $g(x)$. No single-bit error can be caught. In addition, this $g(x)$ is useless because it means the codeword is just the dataword augmented with $n - k$ zeros.

Two Isolated Single-Bit Errors

Now imagine there are two single-bit isolated errors. Under what conditions can this type of error be caught? We can show this type of error as $e(x) = x^j + x^i$. The values of i and j define the positions of the errors, and the difference $j - i$ defines the distance between the two errors, as shown in Figure 10.10.

Figure 10.10 *Representation of two isolated single-bit errors using polynomials*



We can write $e(x) = x^i(x^{j-i} + 1)$. If $g(x)$ has more than one term and one term is x^0, it cannot divide x^i, as we saw in the previous section. So if $g(x)$ is to divide $e(x)$, it must divide $x^{j-i} + 1$. In other words, $g(x)$ must not divide $x^t + 1$, where t is between 0 and $n - 1$. However, $t = 0$ is meaningless and $t = 1$ is needed, as we will see later. This means t should be between 2 and $n - 1$.

If a generator cannot divide $x^t + 1$ (t between 0 and $n - 1$), then all isolated double errors can be detected.

Example 10.9

Find the status of the following generators related to two isolated, single-bit errors.

a. $x + 1$
b. $x^4 + 1$
c. $x^7 + x^6 + 1$
d. $x^{15} + x^{14} + 1$

Solution

a. This is a very poor choice for a generator. Any two errors next to each other cannot be detected.
b. This generator cannot detect two errors that are four positions apart. The two errors can be anywhere, but if their distance is 4, they remain undetected.
c. This is a good choice for this purpose.
d. This polynomial cannot divide any error of type $x^t + 1$ if t is less than 32,768. This means that a codeword with two isolated errors that are next to each other or up to 32,768 bits apart can be detected by this generator.

Odd Numbers of Errors

A generator with a factor of $x + 1$ can catch all odd numbers of errors. This means that we need to make $x + 1$ a factor of any generator. Note that we are not saying that the generator itself should be $x + 1$; we are saying that it should have a factor of $x + 1$. If it is only $x + 1$, it cannot catch the two adjacent isolated errors (see the previous section). For example, $x^4 + x^2 + x + 1$ can catch all odd-numbered errors since it can be written as a product of the two polynomials $x + 1$ and $x^3 + x^2 + 1$.

A generator that contains a factor of $x + 1$ can detect all odd-numbered errors.

Burst Errors

Now let us extend our analysis to the burst error, which is the most important of all. A burst error is of the form $e(x) = (x^j + \cdots + x^i)$. Note the difference between a burst error and two isolated single-bit errors. The first can have two terms or more; the second can only have two terms. We can factor out x^i and write the error as $x^i(x^{j-i} + \cdots + 1)$. If our generator can detect a single error (minimum condition for a generator), then it cannot divide x^i. What we should worry about are those generators that divide $x^{j-i} + \cdots + 1$. In other words, the remainder of $(x^{j-i} + \cdots + 1)/(x^r + \cdots + 1)$ must not be zero. Note that the denominator is the generator polynomial. We can have three cases:

1. If $j - i < r$, the remainder can never be zero. We can write $j - i = L - 1$, where L is the length of the error. So $L - 1 < r$ or $L < r + 1$ or L ð r. This means all burst errors with length smaller than or equal to the number of check bits r will be detected.
2. In some rare cases, if $j - i = r$, or $L = r + 1$, the syndrome is 0 and the error is undetected. It can be proved that in these cases, the probability of undetected burst error of length $r + 1$ is $(1/2)^{r-1}$. For example, if our generator is $x^{14} + x^3 + 1$, in which $r = 14$, a burst error of length $L = 15$ can slip by undetected with the probability of $(1/2)^{14-1}$ or almost 1 in 10,000.
3. In some rare cases, if $j - i > r$, or $L > r + 1$, the syndrome is 0 and the error is undetected. It can be proved that in these cases, the probability of undetected burst error of length greater than $r + 1$ is $(1/2)^r$. For example, if our generator is $x^{14} + x^3 + 1$, in which $r = 14$, a burst error of length greater than 15 can slip by undetected with the probability of $(1/2)^{14}$ or almost 1 in 16,000 cases.

- **All burst errors with $L \le r$ will be detected.**
- **All burst errors with $L = r + 1$ will be detected with probability $1 - (1/2)^{r-1}$.**
- **All burst errors with $L > r + 1$ will be detected with probability $1 - (1/2)^r$.**

Example 10.10

Find the suitability of the following generators in relation to burst errors of different lengths.

a. $x^6 + 1$
b. $x^{18} + x^7 + x + 1$
c. $x^{32} + x^{23} + x^7 + 1$

Solution

a. This generator can detect all burst errors with a length less than or equal to 6 bits; 3 out of 100 burst errors with length 7 will slip by; 16 out of 1000 burst errors of length 8 or more will slip by.
b. This generator can detect all burst errors with a length less than or equal to 18 bits; 8 out of 1 million burst errors with length 19 will slip by; 4 out of 1 million burst errors of length 20 or more will slip by.
c. This generator can detect all burst errors with a length less than or equal to 32 bits; 5 out of 10 billion burst errors with length 33 will slip by; 3 out of 10 billion burst errors of length 34 or more will slip by.

Summary

We can summarize the criteria for a good polynomial generator:

A good polynomial generator needs to have the following characteristics:

1. **It should have at least two terms.**
2. **The coefficient of the term x^0 should be 1.**
3. **It should not divide $x^t + 1$, for t between 2 and $n - 1$.**
4. **It should have the factor $x + 1$.**

Standard Polynomials

Some standard polynomials used by popular protocols for CRC generation are shown in Table 10.4 along with the corresponding bit pattern.

Table 10.4 *Standard polynomials*

Name	*Polynomial*	*Used in*
CRC-8	$x^8 + x^2 + x + 1$ `100000111`	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$ `11000110101`	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$ `10001000000100001`	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ `100000100110000010001110110110111`	LANs

10.3.5 Advantages of Cyclic Codes

We have seen that cyclic codes have a very good performance in detecting single-bit errors, double errors, an odd number of errors, and burst errors. They can easily be implemented in hardware and software. They are especially fast when implemented in hardware. This has made cyclic codes a good candidate for many networks.

10.3.6 Other Cyclic Codes

The cyclic codes we have discussed in this section are very simple. The check bits and syndromes can be calculated by simple algebra. There are, however, more powerful polynomials that are based on abstract algebra involving Galois fields. These are beyond the scope of this book. One of the most interesting of these codes is the **Reed-Solomon code** used today for both detection and correction.

10.3.7 Hardware Implementation

One of the advantages of a cyclic code is that the encoder and decoder can easily and cheaply be implemented in hardware by using a handful of electronic devices. Also, a hardware implementation increases the rate of check bit and syndrome bit calculation. In this section, we try to show, step by step, the process. The section, however, is optional and does not affect the understanding of the rest of the chapter.

Divisor

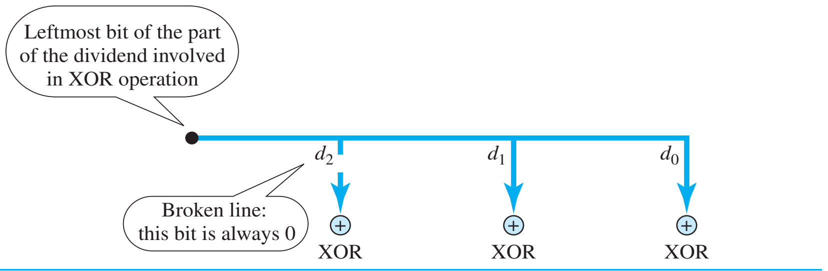
Let us first consider the divisor. We need to note the following points:

1. The divisor is repeatedly XORed with part of the dividend.
2. The divisor has $n - k + 1$ bits which either are predefined or are all 0s. In other words, the bits do not change from one dataword to another. In our previous example, the divisor bits were either 1011 or 0000. The choice was based on the leftmost bit of the part of the augmented data bits that are active in the XOR operation.

3. A close look shows that only $n - k$ bits of the divisor are needed in the XOR operation. The leftmost bit is not needed because the result of the operation is always 0, no matter what the value of this bit. The reason is that the inputs to this XOR operation are either both 0s or both 1s. In our previous example, only 3 bits, not 4, are actually used in the XOR operation.

Using these points, we can make a fixed (hardwired) divisor that can be used for a cyclic code if we know the divisor pattern. Figure 10.11 shows such a design for our previous example. We have also shown the XOR devices used for the operation.

Figure 10.11 *Hardwired design of the divisor in CRC*


Leftmost bit of the part of the dividend involved in XOR operation
d_2
d_1
d_0
Broken line: this bit is always 0
XOR
XOR
XOR

Note that if the leftmost bit of the part of the dividend to be used in this step is 1, the divisor bits ($d_2d_1d_0$) are 011; if the leftmost bit is 0, the divisor bits are 000. The design provides the right choice based on the leftmost bit.

Augmented Dataword

In our paper-and-pencil division process in Figure 10.6, we show the augmented dataword as fixed in position with the divisor bits shifting to the right, 1 bit in each step. The divisor bits are aligned with the appropriate part of the augmented dataword. Now that our divisor is fixed, we need instead to shift the bits of the augmented dataword to the left (opposite direction) to align the divisor bits with the appropriate part. There is no need to store the augmented dataword bits.

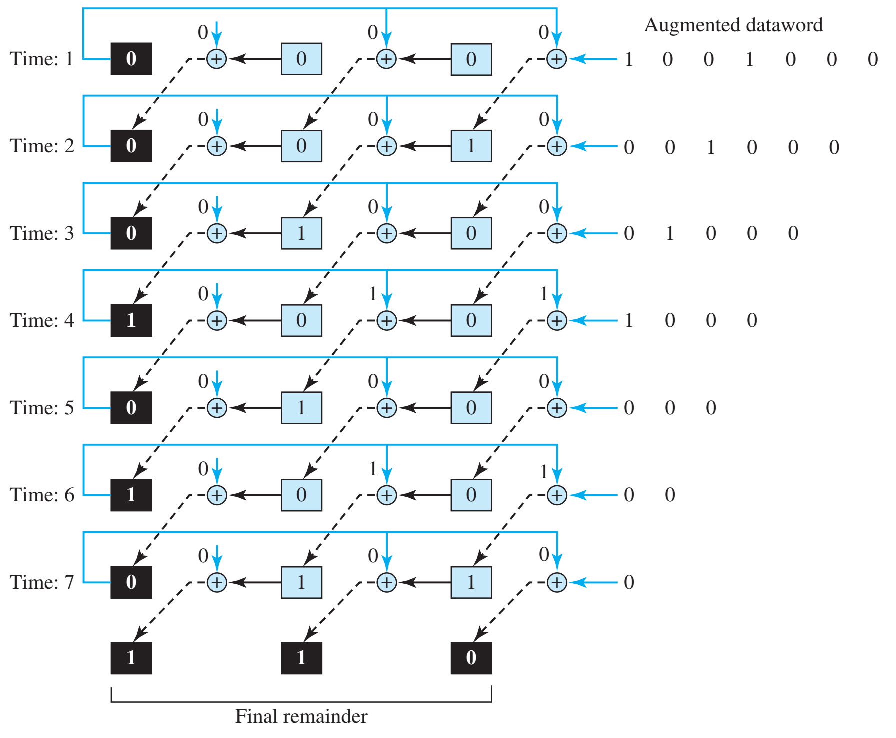
Remainder

In our previous example, the remainder is 3 bits ($n - k$ bits in general) in length. We can use three **registers** (single-bit storage devices) to hold these bits. To find the final remainder of the division, we need to modify our division process. The following is the step-by-step process that can be used to simulate the division process in hardware (or even in software).

1. We assume that the remainder is originally all 0s (000 in our example).
2. At each time click (arrival of 1 bit from an augmented dataword), we repeat the following two actions:
   a. We use the leftmost bit to make a decision about the divisor (011 or 000).
   b. The other 2 bits of the remainder and the next bit from the augmented dataword (total of 3 bits) are XORed with the 3-bit divisor to create the next remainder.

Figure 10.12 shows this simulator, but note that this is not the final design; there will be more improvements.

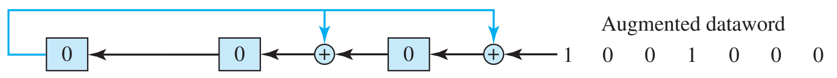
Figure 10.12 *Simulation of division in CRC encoder*



At each clock tick, shown as different times, one of the bits from the augmented dataword is used in the XOR process. If we look carefully at the design, we have seven steps here, while in the paper-and-pencil method we had only four steps. The first three steps have been added here to make each step equal and to make the design for each step the same. Steps 1, 2, and 3 push the first 3 bits to the remainder registers; steps 4, 5, 6, and 7 match the paper-and-pencil design. Note that the values in the remainder register in steps 4 to 7 exactly match the values in the paper-and-pencil design. The final remainder is also the same.

The above design is for demonstration purposes only. It needs simplification to be practical. First, we do not need to keep the intermediate values of the remainder bits; we need only the final bits. We therefore need only 3 registers instead of 24. After the XOR operations, we do not need the bit values of the previous remainder. Also, we do not need 21 XOR devices; two are enough because the output of an XOR operation in which one of the bits is 0 is simply the value of the other bit. This other bit can be used as the output. With these two modifications, the design becomes tremendously simpler and less expensive, as shown in Figure 10.13.

Figure 10.13 *The CRC encoder design using shift registers*

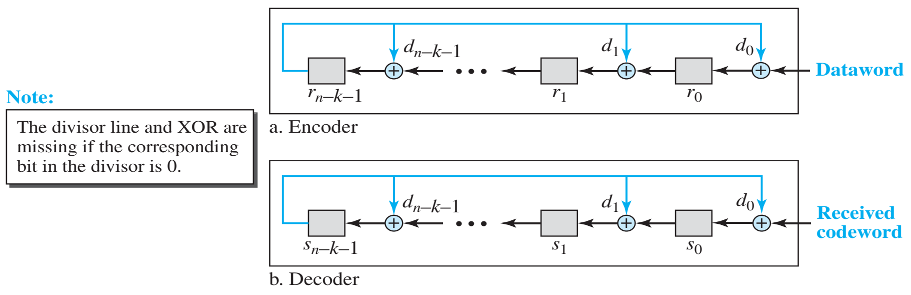


We need, however, to make the registers shift registers. A 1-bit shift register holds a bit for a duration of one clock time. At a time click, the shift register accepts the bit at its input port, stores the new bit, and displays it on the output port. The content and the output remain the same until the next input arrives. When we connect several 1-bit shift registers together, it looks as if the contents of the register are shifting.

General Design

A general design for the encoder and decoder is shown in Figure 10.14.

Figure 10.14 *General design of encoder and decoder of a CRC code*



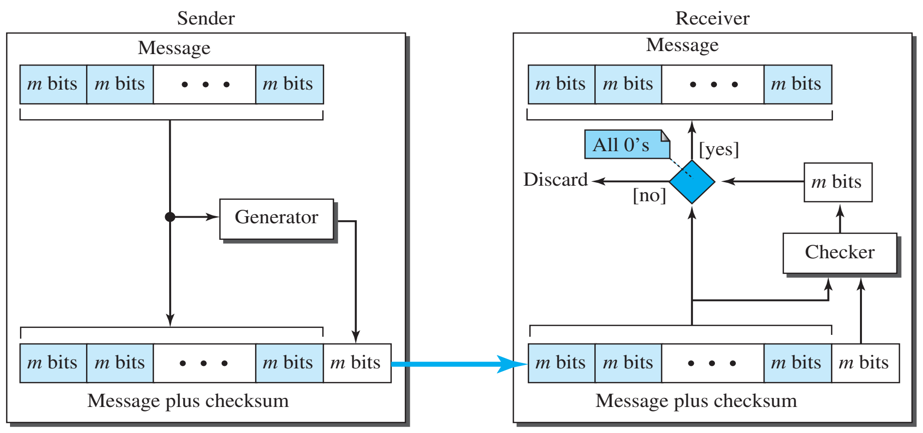
Note that we have $n - k$ 1-bit shift registers in both the encoder and decoder. We have up to $n - k$ XOR devices, but the divisors normally have several 0s in their pattern, which reduces the number of devices. Also note that, instead of augmented datawords, we show the dataword itself as the input because after the bits in the dataword are all fed into the encoder, the extra bits, which all are 0s, do not have any effect on the right-most XOR. Of course, the process needs to be continued for another $n - k$ steps before the check bits are ready. This fact is one of the criticisms of this design. Better schemes have been designed to eliminate this waiting time (the check bits are ready after k steps), but we leave this as a research topic for the reader. In the decoder, however, the entire codeword must be fed to the decoder before the syndrome is ready.

10.4 CHECKSUM

Checksum is an error-detecting technique that can be applied to a message of any length. In the Internet, the checksum technique is mostly used at the network and transport layer rather than the data-link layer. However, to make our discussion of error-detecting techniques complete, we discuss the checksum in this chapter.

At the source, the message is first divided into m-bit units. The generator then creates an extra m-bit unit called the ***checksum,*** which is sent with the message. At the destination, the checker creates a new checksum from the combination of the message and sent checksum. If the new checksum is all 0s, the message is accepted; otherwise, the message is discarded (Figure 10.15). Note that in the real implementation, the checksum unit is not necessarily added at the end of the message; it can be inserted in the middle of the message.

Figure 10.15 *Checksum*



10.4.1 Concept

The idea of the traditional checksum is simple. We show this using a simple example.

Example 10.11

Suppose the message is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the message is not accepted.

One's Complement Addition

The previous example has one major drawback. Each number can be written as a 4-bit word (each is less than 15) except for the sum. One solution is to use **one's complement** arithmetic. In this arithmetic, we can represent unsigned numbers between 0 and $2^m - 1$ using only m bits. If the number has more than m bits, the extra leftmost bits need to be added to the m rightmost bits (wrapping).

Example 10.12

In the previous example, the decimal number 36 in binary is $(100100)_2$. To change it to a 4-bit number we add the extra leftmost bit to the right four bits as shown below.

$$(10)_2 + (0100)_2 = (0110)_2 \rightarrow (6)_{10}$$

Instead of sending 36 as the sum, we can send 6 as the sum (7, 11, 12, 0, 6, **6**). The receiver can add the first five numbers in one's complement arithmetic. If the result is 6, the numbers are accepted; otherwise, they are rejected.

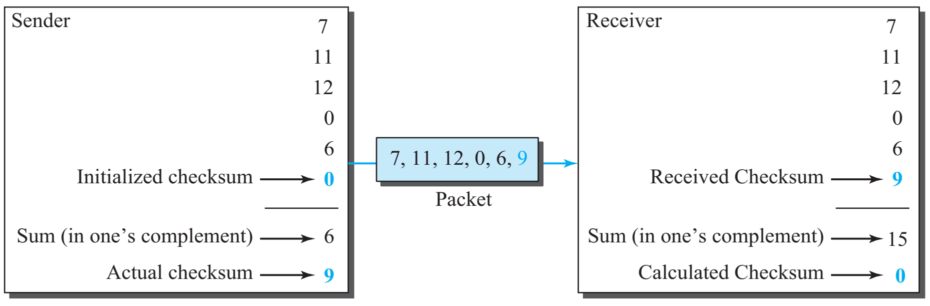
Checksum

We can make the job of the receiver easier if we send the complement of the sum, the checksum. In one's complement arithmetic, the complement of a number is found by completing all bits (changing all 1s to 0s and all 0s to 1s). This is the same as subtracting the number from $2^m - 1$. In one's complement arithmetic, we have two 0s: one positive and one negative, which are complements of each other. The positive zero has all m bits set to 0; the negative zero has all bits set to 1 (it is $2^m - 1$). If we add a number with its complement, we get a negative zero (a number with all bits set to 1). When the receiver adds all five numbers (including the checksum), it gets a negative zero. The receiver can complement the result again to get a positive zero.

Example 10.13

Let us use the idea of the checksum in Example 10.12. The sender adds all five numbers in one's complement to get the sum = 6. The sender then complements the result to get the checksum = 9, which is 15 − 6. Note that $6 = (0110)_2$ and $9 = (1001)_2$; they are complements of each other. The sender sends the five data numbers and the checksum (7, 11, 12, 0, 6, **9**). If there is no corruption in transmission, the receiver receives (7, 11, 12, 0, 6, **9**) and adds them in one's complement to get 15. The sender complements 15 to get 0. This shows that data have not been corrupted. Figure 10.16 shows the process.

Figure 10.16 *Example 10.13*



Internet Checksum

Traditionally, the Internet has used a 16-bit checksum. The sender and the receiver follow the steps depicted in Table 10.5. The sender or the receiver uses five steps.

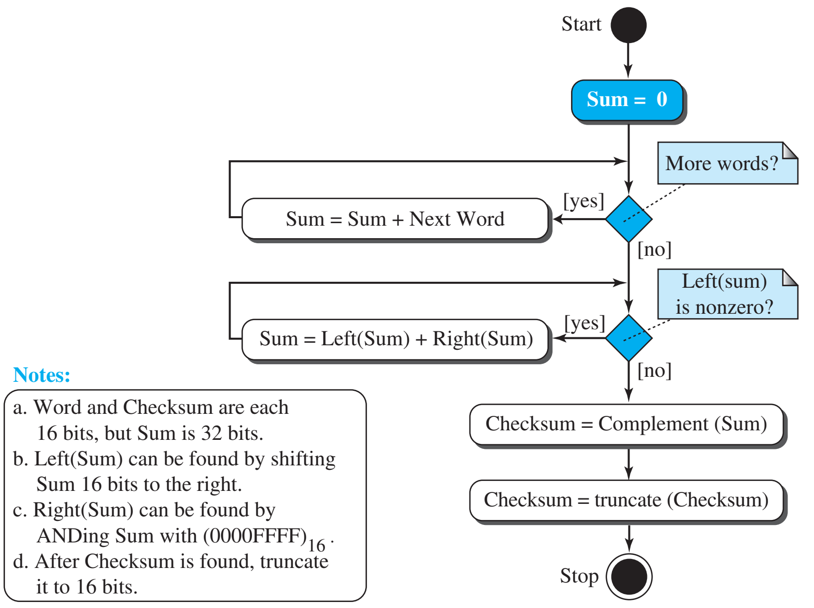
Table 10.5 *Procedure to calculate the traditional checksum*

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum are received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

Algorithm

We can use the flow diagram of Figure 10.17 to show the algorithm for calculation of the checksum. A program in any language can easily be written based on the algorithm. Note that the first loop just calculates the sum of the data units in two's complement; the second loop wraps the extra bits created from the two's complement calculation to simulate the calculations in one's complement. This is needed because almost all computers today do calculation in two's complement.

Figure 10.17 *Algorithm to calculate a traditional checksum*



Notes:

a. Word and Checksum are each 16 bits, but Sum is 32 bits.
b. Left(Sum) can be found by shifting Sum 16 bits to the right.
c. Right(Sum) can be found by ANDing Sum with $(0000FFFF)_{16}$.
d. After Checksum is found, truncate it to 16 bits.

Performance

The traditional checksum uses a small number of bits (16) to detect errors in a message of any size (sometimes thousands of bits). However, it is not as strong as the CRC in error-checking capability. For example, if the value of one word is incremented and the value of another word is decremented by the same amount, the two errors cannot be detected because the sum and checksum remain the same. Also, if the values of several words are incremented but the sum and the checksum do not change, the errors are not detected. Fletcher and Adler have proposed some weighted checksums that eliminate the first problem. However, the tendency in the Internet, particularly in designing new protocols, is to replace the checksum with a CRC.

10.4.2 Other Approaches to the Checksum

As mentioned before, there is one major problem with the traditional checksum calculation. If two 16-bit items are transposed in transmission, the checksum cannot catch this error. The reason is that the traditional checksum is not weighted: it treats each data item equally. In other words, the order of data items is immaterial to the calculation. Several approaches have been used to prevent this problem. We mention two of them here: Fletcher and Adler.

Fletcher Checksum

The Fletcher checksum was devised to weight each data item according to its position. Fletcher has proposed two algorithms: 8-bit and 16-bit. The first, 8-bit Fletcher, calculates on 8-bit data items and creates a 16-bit checksum. The second, 16-bit Fletcher, calculates on 16-bit data items and creates a 32-bit checksum.

The 8-bit Fletcher is calculated over data octets (bytes) and creates a 16-bit checksum. The calculation is done modulo 256 (2^8), which means the intermediate results are divided by 256 and the remainder is kept. The algorithm uses two accumulators, L and R. The first simply adds data items together; the second adds a weight to the calculation. There are many variations of the 8-bit Fletcher algorithm; we show a simple one in Figure 10.18.

The 16-bit Fletcher checksum is similar to the 8-bit Fletcher checksum, but it is calculated over 16-bit data items and creates a 32-bit checksum. The calculation is done modulo 65,536.

Adler Checksum

The Adler checksum is a 32-bit checksum. Figure 10.19 shows a simple algorithm in flowchart form. It is similar to the 16-bit Fletcher with three differences. First, calculation is done on single bytes instead of 2 bytes at a time. Second, the modulus is a prime number (65,521) instead of 65,536. Third, L is initialized to 1 instead of 0. It has been proved that a prime modulo has a better detecting capability in some combinations of data.

Figure 10.18 *Algorithm to calculate an 8-bit Fletcher checksum*

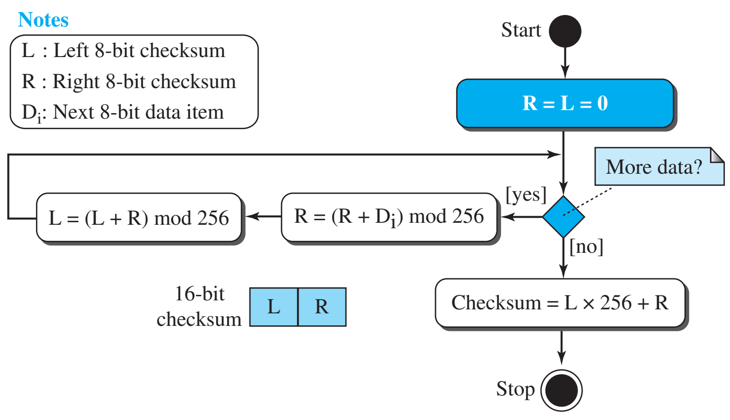
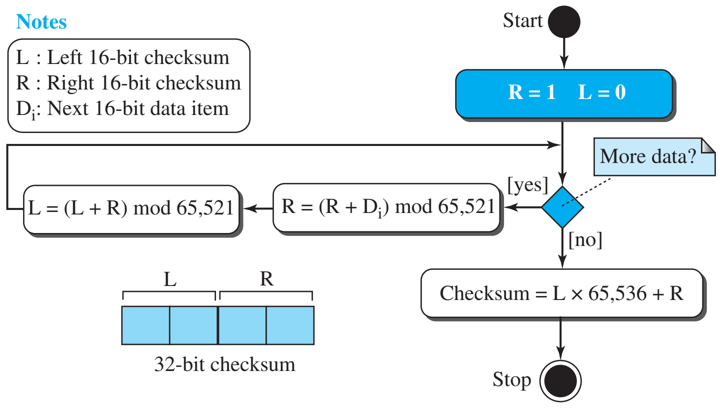


Figure 10.19 *Algorithm to calculate an Adler checksum*



To see the behavior of the different checksum algorithms, check some of the applets for this chapter at the book website.

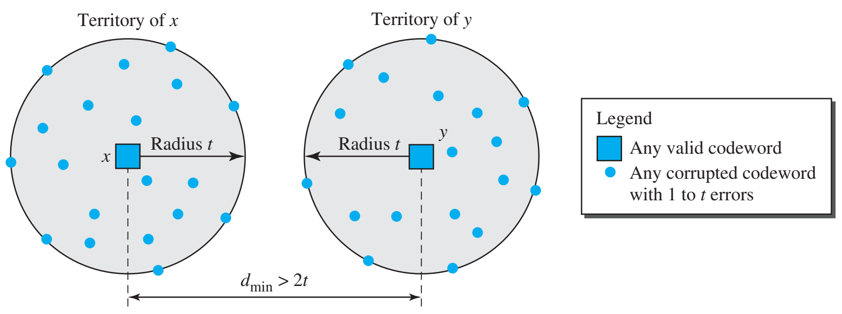
10.5 FORWARD ERROR CORRECTION

We discussed error detection and retransmission in the previous sections. However, retransmission of corrupted and lost packets is not useful for real-time multimedia transmission because it creates an unacceptable delay in reproducing: we need to wait until the lost or corrupted packet is resent. We need to correct the error or reproduce the packet immediately. Several schemes have been designed and used in this case that are collectively referred to as **forward error correction** (**FEC**) techniques. We briefly discuss some of the common techniques here.

10.5.1 Using Hamming Distance

We earlier discussed the Hamming distance for error detection. We said that to detect s errors, the minimum Hamming distance should be $d_{min} = s + 1$. For error detection, we definitely need more distance. It can be shown that to detect t errors, we need to have $d_{min} = 2t + 1$. In other words, if we want to correct 10 bits in a packet, we need to make the minimum hamming distance 21 bits, which means a lot of redundant bits need to be sent with the data. To give an example, consider the famous BCH code. In this code, if data is 99 bits, we need to send 255 bits (extra 156 bits) to correct just 23 possible bit errors. Most of the time we cannot afford such a redundancy. We give some examples of how to calculate the required bits in the practice set. Figure 10.20 shows the geometrical representation of this concept.

Figure 10.20 *Hamming distance for error correction*



10.5.2 Using XOR

Another recommendation is to use the property of the exclusive OR operation as shown below.

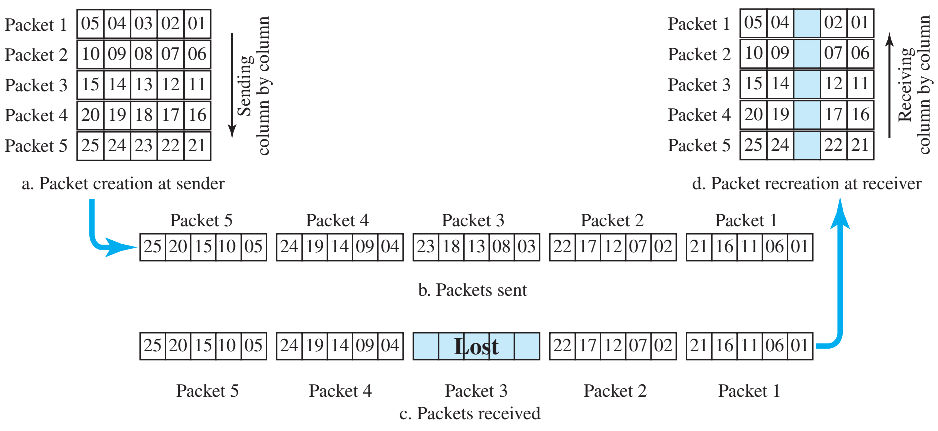
$$\mathbf{R} = \mathbf{P}_1 \oplus \mathbf{P}_2 \oplus \ldots \oplus \mathbf{P}_i \oplus \ldots \oplus \mathbf{P}_N \quad \rightarrow \quad \mathbf{P}_i = \mathbf{P}_1 \oplus \mathbf{P}_2 \oplus \ldots \oplus \mathbf{R} \oplus \ldots \oplus \mathbf{P}_N$$

In other words, if we apply the exclusive OR operation on N data items (P_1 to P_N), we can recreate any of the data items by exclusive-ORing all of the items, replacing the one to be created by the result of the previous operation (R). This means that we can divide a packet into N chunks, create the exclusive OR of all the chunks and send $N + 1$ chunks. If any chunk is lost or corrupted, it can be created at the receiver site. Now the question is what should the value of N be. If $N = 4$, it means that we need to send 25 percent extra data and be able to correct the data if only one out of four chunks is lost.

10.5.3 Chunk Interleaving

Another way to achieve FEC in multimedia is to allow some small chunks to be missing at the receiver. We cannot afford to let all the chunks belonging to the same packet be missing; however, we can afford to let one chunk be missing in each packet. Figure 10.21 shows that we can divide each packet into 5 chunks (normally the number is much larger). We can then create data chunk by chunk (horizontally), but combine the chunks into packets vertically. In this case, each packet sent carries a chunk from several original packets. If the packet is lost, we miss only one chunk in each packet, which is normally acceptable in multimedia communication.

Figure 10.21 *Interleaving*



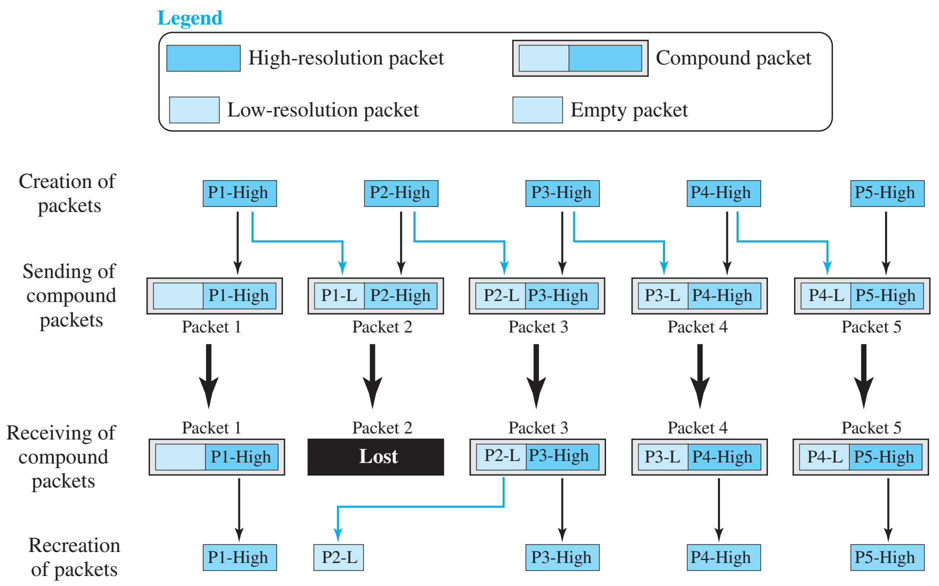
10.5.4 Combining Hamming Distance and Interleaving

Hamming distance and interleaving can be combined. We can first create n-bit packets that can correct t-bit errors. Then we interleave m rows and send the bits column by column. In this way, we can automatically correct burst errors up to $m \times t$-bit errors.

10.5.5 Compounding High- and Low-Resolution Packets

Still another solution is to create a duplicate of each packet with a low-resolution redundancy and combine the redundant version with the next packet. For example, we can create four low-resolution packets out of five high-resolution packets and send them as shown in Figure 10.22. If a packet is lost, we can use the low-resolution version from the next packet. Note that the low-resolution section in the first packet is empty. In this method, if the last packet is lost, it cannot be recovered, but we use the low-resolution version of a packet if the lost packet is not the last one. The audio and video reproduction does not have the same quality, but the lack of quality is not recognized most of the time.

Figure 10.22 *Compounding high- and low-resolution packets*



10.6 END-CHAPTER MATERIALS

10.6.1 Recommended Reading

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items in brackets [. . .] refer to the reference list at the end of the text.

Books

Several excellent books discuss link-layer issues. Among them we recommend [Ham 80], [Zar 02], [Ror 96], [Tan 03], [GW 04], [For 03], [KMK 04], [Sta 04], [Kes 02], [PD 03], [Kei 02], [Spu 00], [KCK 98], [Sau 98], [Izz 00], [Per 00], and [WV 00].

RFCs

A discussion of the use of the checksum in the Internet can be found in RFC 1141.

10.6.2 Key Terms

block coding
burst error
check bit
checksum
codeword
convolution coding
cyclic code
cyclic redundancy check (CRC)
dataword
error
error correction
error detection
forward error correction (FEC)
generator polynomial
Hamming distance
interference
linear block code
minimum Hamming distance
modular arithmetic
one's complement
parity-check code
polynomial
redundancy
register
Reed-Solomon code
single-bit error
syndrome

10.6.3 Summary

Data can be corrupted during transmission. Some applications require that errors be detected and corrected. In a single-bit error, only one bit in the data unit has changed. A burst error means that two or more bits in the data unit have changed. To detect or correct errors, we need to send extra (redundant) bits with data. There are two main methods of error correction: forward error correction and correction by retransmission.

We can divide coding schemes into two broad categories: block coding and convolution coding. In coding, we need to use modulo-2 arithmetic. Operations in this arithmetic are very simple; addition and subtraction give the same results. We use the XOR (exclusive OR) operation for both addition and subtraction. In block coding, we divide our message into blocks, each of k bits, called *datawords*. We add r redundant bits to each block to make the length $n = k + r$. The resulting n-bit blocks are called *codewords*.

In block coding, errors be detected by using the following two conditions:

a. The receiver has (or can find) a list of valid codewords.

b. The original codeword has changed to an invalid one.

The Hamming distance between two words is the number of differences between corresponding bits. The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words. To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{\min} = s + 1$. To guarantee correction of up to t errors in all cases, the minimum Hamming distance in a block code must be $d_{\min} = 2t + 1$.

In a linear block code, the exclusive OR (XOR) of any two valid codewords creates another valid codeword.

A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{\min} = 2$. A simple parity-check code can detect an odd number of errors.

All Hamming codes discussed in this book have $d_{\min} = 3$. The relationship between m and n in these codes is $n = 2m - 1$.

Cyclic codes are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword. A category of cyclic codes called the cyclic redundancy check (CRC) is used in networks such as LANs and WANs.

A pattern of 0s and 1s can be represented as a polynomial with coefficients of 0 and 1. Traditionally, the Internet has been using a 16-bit checksum, which uses one's complement arithmetic. In this arithmetic, we can represent unsigned numbers between 0 and $2^n - 1$ using only n bits.

10.7 PRACTICE SET

10.7.1 Quizzes

A set of interactive quizzes for this chapter can be found on the book website. It is strongly recommended that the student take the quizzes to check his/her understanding of the materials before continuing with the practice set.

10.7.2 Questions

Q10-1. How does a single-bit error differ from a burst error?

Q10-2. What is the definition of a linear block code?

Q10-3. In a block code, a dataword is 20 bits and the corresponding codeword is 25 bits. What are the values of k, r, and n according to the definitions in the text? How many redundant bits are added to each dataword?

Q10-4. In a codeword, we add two redundant bits to each 8-bit data word. Find the number of

a. valid codewords. **b.** invalid codewords.

Q10-5. What is the minimum Hamming distance?

Q10-6. If we want to be able to detect two-bit errors, what should be the minimum Hamming distance?

Q10-7. A category of error detecting (and correcting) code, called the ***Hamming code,*** is a code in which $d_{min} = 3$. This code can detect up to two errors (or correct one single error). In this code, the values of n, k, and r are related as: $n = 2^r - 1$ and $k = n - r$. Find the number of bits in the dataword and the codewords if r is 3.

Q10-8. In CRC, if the dataword is 5 bits and the codeword is 8 bits, how many 0s need to be added to the dataword to make the dividend? What is the size of the remainder? What is the size of the divisor?

Q10-9. In CRC, which of the following generators (divisors) guarantees the detection of a single bit error?

a. 101 **b.** 100 **c.** 1

Q10-10. In CRC, which of the following generators (divisors) guarantees the detection of an odd number of errors?

a. 10111 **b.** 101101 **c.** 111

Q10-11. In CRC, we have chosen the generator 1100101. What is the probability of detecting a burst error of length

a. 5? **b.** 7? **c.** 10?

Q10-12. Assume we are sending data items of 16-bit length. If two data items are swapped during transmission, can the traditional checksum detect this error? Explain.

Q10-13. Can the value of a traditional checksum be all 0s (in binary)? Defend your answer.

Q10-14. Show how the Fletcher algorithm (Figure 10.18) attaches weights to the data items when calculating the checksum.

Q10-15. Show how the Adler algorithm (Figure 10.19) attaches weights to the data items when calculating the checksum.

10.7.3 Problems

P10-1. What is the maximum effect of a 2-ms burst of noise on data transmitted at the following rates?

a. 1500 bps **b.** 12 kbps **c.** 100 kbps **d.** 100 Mbps

P10-2. Assume that the probability that a bit in a data unit is corrupted during transmission is p. Find the probability that x number of bits are corrupted in an n-bit data unit for each of the following cases.

a. $n = 8, x = 1, p = 0.2$

b. $n = 16, x = 3, p = 0.3$

c. $n = 32, x = 10, p = 0.4$

P10-3. Exclusive-OR (XOR) is one of the most used operations in the calculation of codewords. Apply the exclusive-OR operation on the following pairs of patterns. Interpret the results.

a. (10001) $\oplus$ (10001) **b.** (11100) $\oplus$ (00000) **c.** (10011) $\oplus$ (11111)

P10-4. In Table 10.1, the sender sends dataword 10. A 3-bit burst error corrupts the codeword. Can the receiver detect the error? Defend your answer.

P10-5. Using the code in Table 10.2, what is the dataword if each of the following codewords is received?

a. 01011 **b.** 11111 **c.** 00000 **d.** 11011

P10-6. Prove that the code represented by the following codewords is not linear. You need to find only one case that violates the linearity.

{(00000), (01011), (10111), (11111)}

P10-7. What is the Hamming distance for each of the following codewords?

a. d (10000, 00000) **b.** d (10101, 10000)
c. d (00000, 11111) **d.** d (00000, 00000)

P10-8. Although it can be formally proved that the code in Table 10.3 is both linear and cyclic, use only two tests to partially prove the fact:

a. Test the cyclic property on codeword 0101100.

b. Test the linear property on codewords 0010110 and 1111111.

P10-9. Referring to the CRC-8 in Table 5.4, answer the following questions:

a. Does it detect a single error? Defend your answer.

b. Does it detect a burst error of size 6? Defend your answer.

c. What is the probability of detecting a burst error of size 9?

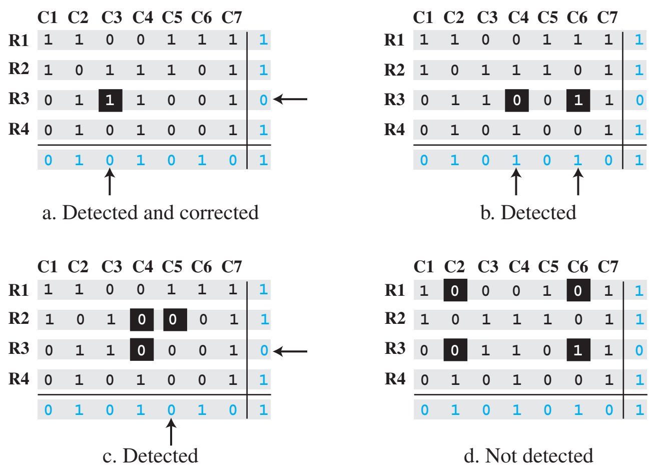
d. What is the probability of detecting a burst error of size 15?

P10-10. Assuming even parity, find the parity bit for each of the following data units.

a. 1001011 **b.** 0001100 **c.** 1000000 **d.** 1110111

P10-11. A simple parity-check bit, which is normally added at the end of the word (changing a 7-bit ASCII character to a byte), cannot detect even numbers of errors. For example, two, four, six, or eight errors cannot be detected in this way. A better solution is to organize the characters in a table and create row and column parities. The bit in the row parity is sent with the byte, the column parity is sent as an extra byte (Figure 10.23).

Figure 10.23 *P10-11*



Show how the following errors can be detected:

a. An error at (R3, C3).

b. Two errors at (R3, C4) and (R3, C6).

c. Three errors at (R2, C4), (R2, C5), and (R3, C4).

d. Four errors at (R1, C2), (R1, C6), (R3, C2), and (R3, C6).

P10-12. Given the dataword 101001111 and the divisor 10111, show the generation of the CRC codeword at the sender site (using binary division).

P10-13. Apply the following operations on the corresponding polynomials:

- **a.** $(x^3 + x^2 + x + 1) + (x^4 + x^2 + x + 1)$
- **b.** $(x^3 + x^2 + x + 1) - (x^4 + x^2 + x + 1)$
- **c.** $(x^3 + x^2) \times (x^4 + x^2 + x + 1)$
- **d.** $(x^3 + x^2 + x + 1) / (x^2 + 1)$

P10-14. Answer the following questions:

- **a.** What is the polynomial representation of 101110?
- **b.** What is the result of shifting 101110 three bits to the left?
- **c.** Repeat part b using polynomials.
- **d.** What is the result of shifting 101110 four bits to the right?
- **e.** Repeat part d using polynomials.

P10-15. Which of the following CRC generators guarantee the detection of a single bit error?

a. $x^3 + x + 1$ **b.** $x^4 + x^2$ **c.** 1 **d.** $x^2 + 1$

P10-16. Referring to the CRC-8 polynomial in Table 10.7, answer the following questions:

- **a.** Does it detect a single error? Defend your answer.
- **b.** Does it detect a burst error of size 6? Defend your answer.
- **c.** What is the probability of detecting a burst error of size 9?
- **d.** What is the probability of detecting a burst error of size 15?

P10-17. Referring to the CRC-32 polynomial in Table 10.4, answer the following questions:

- **a.** Does it detect a single error? Defend your answer.
- **b.** Does it detect a burst error of size 16? Defend your answer.
- **c.** What is the probability of detecting a burst error of size 33?
- **d.** What is the probability of detecting a burst error of size 55?

P10-18. Assume a packet is made only of four 16-bit words $(A7A2)_{16}$, $(CABF)_{16}$, $(903A)_{16}$, and $(A123)_{16}$. Manually simulate the algorithm in Figure 10.17 to find the checksum.

P10-19. Traditional checksum calculation needs to be done in one's complement arithmetic. Computers and calculators today are designed to do calculations in two's complement arithmetic. One way to calculate the traditional checksum is to add the numbers in two's complement arithmetic, find the quotient and remainder of dividing the result by 2^{16}, and add the quotient and the remainder to get the sum in one's complement. The checksum can be found by subtracting the sum from $2^{16} - 1$. Use the above method to find the checksum of the following four numbers: 43,689, 64,463, 45,112, and 59,683.

P10-20. This problem shows a special case in checksum handling. A sender has two data items to send: $(4567)_{16}$ and $(BA98)_{16}$. What is the value of the checksum?

P10-21. Manually simulate the Fletcher algorithm (Figure 10.18) to calculate the checksum of the following bytes: $(2B)_{16}$, $(3F)_{16}$, $(6A)_{16}$, and $(AF)_{16}$. Also show that the result is a weighted checksum.

P10-22. Manually simulate the Adler algorithm (Figure 10.19) to calculate the checksum of the following words: $(FBFF)_{16}$ and $(EFAA)_{16}$. Also show that the result is a weighted checksum.

P10-23. One of the examples of a weighted checksum is the ISBN-10 code we see printed on the back cover of some books. In ISBN-10, there are 9 decimal digits that define the country, the publisher, and the book. The tenth (rightmost) digit is a checksum digit. The code, $D_1D_2D_3D_4D_5D_6D_7D_8D_9C$, satisfies the following.

$$[(10 \times D_1) + (9 \times D_2) + (8 \times D_3) + \ldots + (2 \times D_9) + (1 \times C)] \bmod 11 = 0$$

In other words, the weights are 10, 9, . . .,1. If the calculated value for C is 10, one uses the letter X instead. By replacing each weight w with its complement in modulo 11 arithmetic $(11 - w)$, it can be shown that the check digit can be calculated as shown below.

$$C = [(1 \times D_1) + (2 \times D_2) + (3 \times D_3) + \ldots + (9 \times D_9)] \bmod 11$$

Calculate the check digit for ISBN-10: **0-07-296775-C**.

P10-24. An ISBN-13 code, a new version of ISBN-10, is another example of a weighted checksum with 13 digits, in which there are 12 decimal digits defining the book and the last digit is the checksum digit. The code, $D_1D_2D_3D_4D_5D_6D_7D_8D_9D_{10}D_{11}D_{12}C$, satisfies the following.

$$[(1 \times D_1) + (3 \times D_2) + (1 \times D_3) + \ldots + (3 \times D_{12}) + (1 \times C)] \bmod 10 = 0$$

In other words, the weights are 1 and 3 alternately. Using the above description, calculate the check digit for ISBN-13: **978-0-07-296775-C**.

P10-25. In the interleaving approach to FEC, assume each packet contains 10 samples from a sampled piece of music. Instead of loading the first packet with the first 10 samples, the second packet with the second 10 samples, and so on, the sender loads the first packet with the odd-numbered samples of the first 20 samples, the second packet with the even-numbered samples of the first 20 samples, and so on. The receiver reorders the samples and plays them. Now assume that the third packet is lost in transmission. What will be missed at the receiver site?

P10-26. Assume we want to send a dataword of two bits using FEC based on the Hamming distance. Show how the following list of datawords/codewords can automatically correct up to a one-bit error in transmission.

$$00 \rightarrow 00000 \qquad 01 \rightarrow 01011 \qquad 10 \rightarrow 10101 \qquad 11 \rightarrow 11110$$

P10-27. Assume we need to create codewords that can automatically correct a one-bit error. What should the number of redundant bits (r) be, given the number of bits in the dataword (k)? Remember that the codeword needs to be $n = k + r$ bits, called $C(n, k)$. After finding the relationship, find the number of bits in r if k is 1, 2, 5, 50, or 1000.

P10-28. In the previous problem we tried to find the number of bits to be added to a dataword to correct a single-bit error. If we need to correct more than one bit, the number of redundant bits increases. What should the number of redundant bits (r) be to automatically correct one or two bits (not necessarily contiguous) in a dataword of size k? After finding the relationship, find the number of bits in r if k is 1, 2, 5, 50, or 1000.

P10-29. Using the ideas in the previous two problems, we can create a general formula for correcting any number of errors (m) in a codeword of size (n). Develop such a formula. Use the combination of n objects taking x objects at a time.

P10-30. In Figure 10.22, assume we have 100 packets. We have created two sets of packets with high and low resolutions. Each high-resolution packet carries on average 700 bits. Each low-resolution packet carries on average 400 bits. How many extra bits are we sending in this scheme for the sake of FEC? What is the percentage of overhead?

10.8 SIMULATION EXPERIMENTS

10.8.1 Applets

We have created some Java applets to show some of the main concepts discussed in this chapter. It is strongly recommended that the students activate these applets on the book website and carefully examine the protocols in action.

10.9 PROGRAMMING ASSIGNMENTS

For each of the following assignments, write a program in the programming language you are familiar with.

Prg10-1. A program to simulate the calculation of CRC.

Prg10-2. A program to simulate the calculation of traditional checksum.

Prg10-3. A program to simulate the calculation of Fletcher checksum.

Prg10-4. A program to simulate the calculation of Adler checksum.