

DATA COMMUNICATIONS CONCEPTS

Concepts Reinforced

I-P-O Model

Protocols & compatibility

Concepts Introduced

Data digitization
Half-duplex vs. full-duplex
Data compression
Modulation techniques
Packetization
Error detection and correction
Circuit vs. packet switching
Serial vs. parallel communications

Analog vs. digital communications
Synchronous vs. asynchronous
transmission
Baud rate vs. transmission rate
Multiplexing
Flow control
Connectionless vs. connection-
oriented networks

OBJECTIVES

Upon successful completion of this chapter, you should be able to:

1. Distinguish between the following related concepts and understand the proper application of each of the following terms:
 - analog
 - synchronous
 - full duplex
 - serial
 - bps
 - error detection
 - circuit switching
 - connectionless
 - digital
 - asynchronous
 - half duplex
 - parallel
 - baud rate
 - error correction
 - packet switching
 - connection-oriented
2. Understand the concepts, processes and protocols involved with completing an end-to-end data communications session, including the following:
 - character encoding
 - modulation/demodulation
 - data switching

3. Understand the impact and limitations of various modulation techniques.
4. Understand the impact and limitations of various multiplexing techniques.

■ INTRODUCTION

This chapter introduces the basic concepts and vocabulary of data communications. The concepts and terms introduced in this chapter form the foundation for the data communication technologies introduced later in the book so it is critical to develop a thorough understanding of the material before reading further. Discussions of specific technology will be kept to a minimum in this chapter, concentrating instead on the conceptual aspects of the “how” of data transmission. The in-depth study of communications technology, including its business impact, will begin in Chapter 3.

Overall Data Communication Architecture

Figure 2–1 serves as a roadmap for all of the concepts to be studied in the chapter. An end-to-end communication session between two computers across the Internet offers a framework into which most data communication concepts can be introduced. Each concept will be introduced individually and then explained in terms of its contribution to the overall end-to-end communication.

In this chapter, basic data communications concepts are introduced in a generic manner. These concepts are implemented in specific data communications technologies in different ways, depending on the environment the technology is intended to operate. Multiple technologies including modems, cable modems, and digital subscriber line (DSL) technologies can be used as the actual mechanism for gaining access to the Internet. These technologies, along with the manner in which they implement the basic data communication concepts, will be discussed in chapter 3.

This chapter will focus on the basic data communications concepts used in each of the areas defined in Figure 2–1. The technologies used to implement each portion of the overall end-to-end communication link will be introduced in the following chapters:

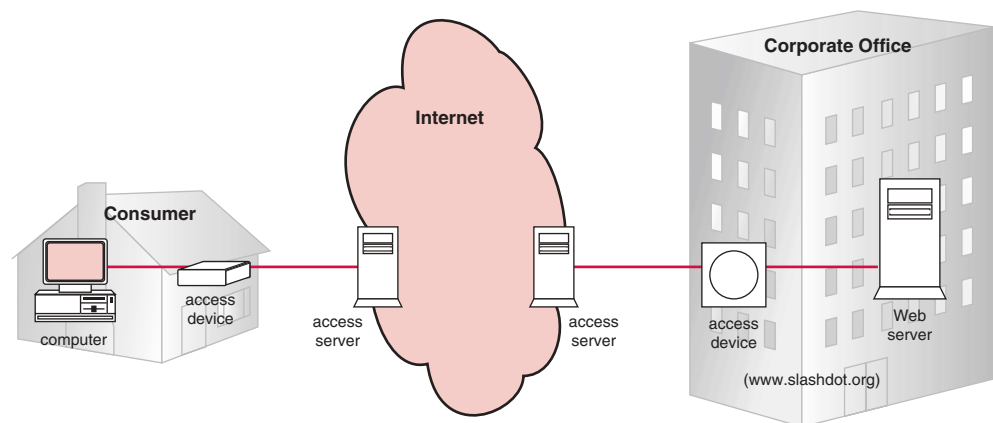


Figure 2–1 Basic Data Communications Concepts Used to Access the Internet

- PC to access device and access device to access server —Chapter 3
- Local area networks—Chapter 4
- Access device to access device across the Internet —Chapter 7
- Network protocols used end-to-end—Chapters 5 & 6

■ DATA DIGITIZATION

Before it can be sent across a data network, information must be converted from its native human interpretable form into a format native to computers. Humans operate in an analog world. Sounds have varying frequency and loudness, pictures have varying colors and shapes, and each character of text can be one of several options, depending on the language being used.

Computers operate in a digital world: the only values they understand are one and zero. All data must be represented as a series of these two basic values. Each one or zero in the data stream is known as **bit**. The digital data-bit stream is usually broken into chunks of 8 bits, known as a **byte or octet**. The significance of 8 bits to a byte is based on the number of bits required to represent a single character in the most common encoding schemes discussed later in this section.

The process of converting analog human interpretable data into digital data for use by computers is known as *digitization*. There are many different techniques available to digitize data. For audio and video information, specialized computer programs known as CODECS (COder/DECoder) are used. Examples of data streams produced by CODECS include MP3, AAC, and DIVX. In addition to simply providing a means of converting analog data to a digital format, most CODECS provide a means of compressing the data. This compression happens prior to the data entering the data communication system and should not be confused with online data compression techniques discussed later in this chapter. Although a thorough discussion of CODECS and the data conversion algorithms they employ is beyond the scope of this book, their importance to modern data communications systems cannot be overestimated.

For textual information, the process of transforming humanly readable characters into machine-readable code is known as **character encoding**. Using an encoding scheme, characters are turned into a series of ones and zeroes. There are multiple protocols or standards that can be used to code characters. The most commonly used standards include ASCII, EBCDIC, and UNICODE.

ASCII

American Standard Code for Information Interchange (ASCII) is one standardized method for encoding humanly readable characters. Standardized by ANSI, ASCII uses a series of 7 bits to represent 128 ($2^7 = 128$) different characters, including uppercase and lowercase letters, numerals, punctuation and symbols, and specialized control characters. One use of control characters will be explained further in Chapter 4. An eighth bit, known as a parity bit, is added to 7-bit ASCII for error detection. Error detection and parity checking will also be described in Chapter 4. Figure 2-2 is an ASCII table that illustrates the relationship between characters and their associated ASCII codes.

MSB				LSB														
Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 3	Bit 2	Bit 1	Bit 0								
0	0	0	0	0	1	1	1	1	1	1	NUL	DLE	SP	0	@	P		p
0	0	1	1	0	0	0	1	0	0	0	SOH	DC1	!	1	A	Q	a	q
0	1	0	0	0	0	0	0	1	0	0	STX	DC2		2	B	R	b	r
1	1	0	0	0	0	0	1	1	0	0	ETX	DC3	#	3	C	S	c	s
0	0	1	0	0	0	0	0	0	1	0	EOT	DC4	\$	4	D	T	d	t
1	0	1	0	0	0	0	1	0	1	0	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	0	0	0	0	1	1	0	ACK	SYN	&	6	F	V	f	v
1	1	1	0	0	0	0	1	1	1	0	BEL	ETB		7	G	W	g	w
0	0	0	1	0	0	0	1	0	0	1	BS	CAN	(8	H	X	h	x
1	0	0	1	0	0	0	1	0	0	1	HT	EM)	9	I	Y	i	y
0	1	0	1	0	0	0	1	0	0	1	LF	SUB	*	:	J	Z	j	z
1	1	0	1	0	0	0	1	0	0	1	VT	ESC	+	;	K	[k	{
0	0	1	1	0	0	0	1	0	0	1	FF	FS	,	<	L	\	l	
1	0	1	1	0	0	0	1	0	0	1	CR	GS		=	M]	m	}
0	1	1	1	0	0	0	1	0	0	1	SO	RS	.	>	N	^	n	~
1	1	1	1	0	0	0	1	0	0	1	SI	US		?	O	-	o	DEL

Figure 2-2 ASCII Table

EBCDIC

Extended Binary Coded Decimal Interchange Code (EBCDIC) is an IBM proprietary 8-bit code capable of representing 256 different characters, numerals, and control characters ($2^8 = 256$). EBCDIC is the primary coding method used in IBM mainframe applications. Figure 2-3 is an EBCDIC table that illustrates the relationship between characters and their associated EBCDIC codes.



Practical Advice
and Information

USING ASCII AND EBCDIC TABLES

Using ASCII or EBCDIC tables to interpret character encoding is relatively straightforward. The tables are arranged according to groups of bits otherwise known as *bit patterns*. The bit patterns are divided into groups. In the case of ASCII, bits 6 through 4 are known as the most significant bits (MSB) while bits 3 through 0 are known as the least significant bits (LSB). In the case of EBCDIC, bits 0 through 3 are known as the MSB and bits 4 through 7 are known as the LSB.

In order to find the bit pattern of a particular character, one needs to just combine the bit patterns that intersect in the table at the character in question, remembering that most significant bits always come before least significant bits. In the case of

MSB	Bit 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	Bit 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	Bit 2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	Bit 3	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

LSB																			
Bit 7	Bit 6	Bit 5	Bit 4																
0	0	0	0	NUL	DLE	DS		SP	&	-							0		
1	0	0	0	SOH	DC1	SOS					a	j			A	J	1		
0	1	0	0	STX	DC2	FS	SYN				b	k	s		B	K	S	2	
1	1	0	0	ETX	DC3						c	l	t		C	L	T	3	
0	0	1	0	PF	RES	BYP	PN				d	m	u		D	M	U	4	
1	0	1	0	HT	NL	LF	RS				e	n	v		E	N	V	5	
0	1	1	0	LC	BS	EOB	UC				f	o	w		F	O	W	6	
1	1	1	0	DEL	IL	PRE	EOT				g	p	x		G	P	X	7	
0	0	0	1		CAN						h	q	y		H	Q	Y	8	
1	0	0	1		EM						\	i	r	z		I	R	Z	9
0	1	0	1	SMM	CC	SM		>>	!	:									
1	1	0	1	VT				.	\$,	#								
0	0	1	1	FF	IFS		DC4	<	*	%	@								
1	0	1	1	CR	IGS	ENQ	NAK	()										
0	1	1	1	SO	IRS	ACK		+	;	>	=								
1	1	1	1	SI	IUS	BEL	SUB		-	?									

Figure 2-3 EBCDIC Table

ASCII, this means that bits are arranged from bit 6 to bit 0, while EBCDIC is arranged from bit 0 to bit 7. As an example, representative characters, numerals, and control characters and their bit patterns are highlighted with shading in the ASCII and EBCDIC tables and are displayed in Figure 2-4 in humanly readable, ASCII, and EBCDIC formats.

Humanly Readable	ASCII	EBCDIC
A	1000001	11000001
x	1111000	10100111
5	0110101	11110101
LF (Line Feed)	0001010	00100101

Figure 2-4 Human readable characters and their corresponding ASCII and EBCDIC codes

UNICODE and ISO 10646

ASCII and EBCDIC coding schemes have sufficient capacity to represent letters and characters familiar to people whose alphabets use the letters A, B, C, and so on. However, what happens if the computer needs to support communication in Cantonese or Arabic? It should be obvious that 128 or 256 possible characters will not suffice when other languages and alphabets are considered.

To resolve this issue, two different efforts were undertaken to establish a new coding standard that could support many more alphabets and symbols than ASCII or EBCDIC. The efforts were ultimately combined and a single standard, **ISO (International Standards Organization) 10646**, also known as **UNICODE** Version 1.1 was released in 1993.

UNICODE is a 16-bit code supporting up to 65,536 possible characters ($2^{16} = 65,536$). It is backward compatible with ASCII because the first 128 characters are identical to the ASCII table. In addition, UNICODE includes more than 2,000 Han characters for languages such as Chinese, Japanese, and Korean. It also includes Hebrew, Greek, Russian, and Sanskrit alphabets as well as mathematical and technical symbols, publishing symbols, geometric shapes, and punctuation marks.

Application programs that display text on a monitor must encode characters according to an encoding scheme understood by the computer's operating system. It is up to the operating systems vendors to include support for particular encoding schemes such as UNICODE/ISO 10646. Windows NT, 2000, and XP, along with most major UNIX variants, support UNICODE.

■ DATA TRANSMISSION TECHNIQUES

There are several concepts that apply to all data transmission technologies. This section introduces these key building blocks that will be used throughout the book to define data transmission technologies.

Serial vs. Parallel Transmission

The bits that represent human-readable characters can be transmitted in either of two basic transmission methodologies: either simultaneously (**parallel transmission**) or in a linear fashion, one after the other (**serial transmission**). The difference between each of these transmission methodologies is illustrated in Figure 2-5.

As illustrated in Figure 2-5, parallel communication is natively faster than serial communication as multiple bits are sent concurrently. There is a key limiting factor to the application of parallel communication, however: The multiple signal lines running in parallel tend to create interference with each other in an electrical transmission environment. The changing electrical signal on one wire creates a magnetic field that induces a ghost copy of the signal onto the surrounding wires. When multiple lines are running in parallel, this tends to quickly muddle the signal. This phenomenon is worsened by transmission speed and distance, thus limiting the application of parallel transmission to fairly short distances.

A common application for parallel communication is communicating between subsystems within a computer. In this application, distance is limited and the system designers can include many ground lines along the bus to bleed off any induced

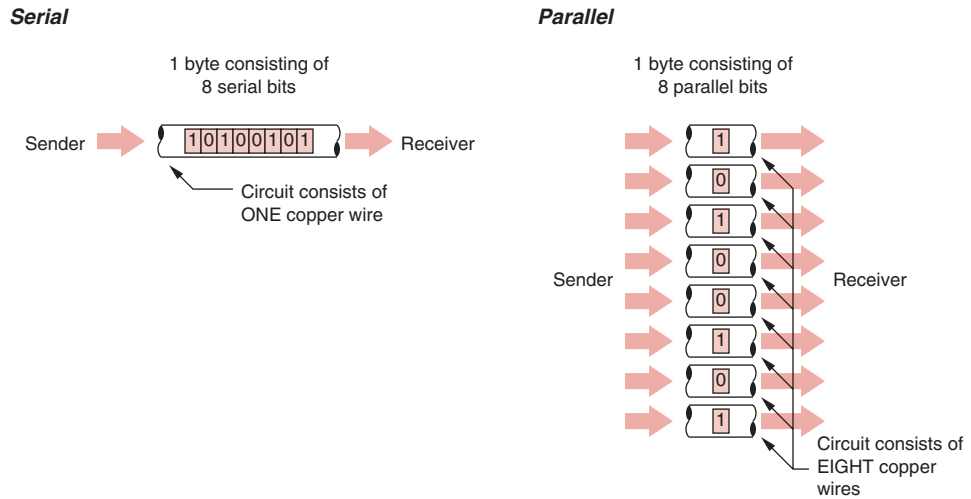


Figure 2-5 Serial and Parallel Transmission Illustrated

voltage, thus increasing signal quality. The term bus is often used to refer to a parallel communication channel in this application as in process bus, memory bus, or expansion bus. Common examples of this type of parallel communication include the PCI bus and IDE/ATA and SCSI hard drive connections.

Serial communication is typically used between computers and external devices. In this application the reduced number of wires required to carry the signal means that cables are less expensive and more flexible. For this reason, all local area and wide area network connections are serial in nature. As electronic signaling and sensing components have increased in speed and sensitivity modern serial standards can operate at speeds that can surpass that of their parallel counterparts allowing high-speed serial technologies such as serial ATA to replace older parallel based technologies.

The advantages, limitations and typical applications of parallel and serial transmission methodologies are summarized in Figure 2-6. Most data-communication technologies use serial transmission techniques, all technologies throughout the book can be assumed to use serial communication techniques unless otherwise stated.

Transmission Characteristic	Serial	Parallel
Transmission Description	One bit comes after another, one at a time.	All bits in a single character are transmitted simultaneously.
Comparative Speed	Slower	Faster
Distance Limitation	Farther	Shorter
Application	Between two computers, from computers to external devices, local and wide area networks	Within a computer along the computer's busses, between a drive controller and a hard drive
Cable Description	All bits travel down a single wire, one bit at a time.	Each bit travels down its own wire simultaneously with other bits.

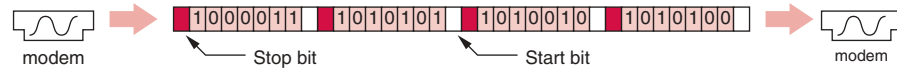
Figure 2-6 Serial Transmission vs. Parallel Transmission Summary

Synchronous vs. Asynchronous Transmission

When devices are communicating, they are exchanging some sort of detectable signal that represents the data. Unless the devices are so busy that they never stop talking, there will be a time when no signal is being sent across the communication channel. During these “dead” times there is no data being sent across the channel. Later when there is additional data to be sent the sending device will begin signaling again. When this happens there must be a way for the destination device to know when to begin looking for data: The two devices must establish and maintain some type of timing between them so that signals are produced, transmitted, and detected accurately. There are two main alternatives to establishing and maintaining the timing for the sampling of the signals. These two timing alternatives are known as **asynchronous** and **synchronous** transmission.

Figure 2-7 summarizes some important characteristics about asynchronous and synchronous transmission methods for 8-bit character-based data transmission. The most noticeable difference is the manner in which signal synchronization is maintained. In asynchronous communication the synchronization is reestablished with the transmission of each character via the use of start and stop bits. Depending on the technology used there may be 1, 1.5, or 2 stop bits. In synchronous communication the synchronization is maintained by a special synchronization bit in each block of data provided by a clocking signal supplied by the communication devices or the signal carrier. Due to the fewer number of bits used to maintain synchronization in synchronous communication; it is more efficient than asynchronous communications.

Asynchronous transmission



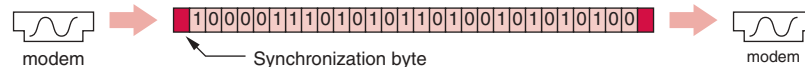
Characteristics:

- Data is sent one byte at a time
- Each byte has a start and 1, 1.5, or 2 stop bits
- Synchronization is reestablished for each byte
- Time between bytes is unsynchronized and of random length

Efficiency (1000 byte transmission)

Control / overhead bits: 1 start and stop bits per byte
 2 control bits per byte x 1000 bytes = 2000 control bits
 7000 data bits / 9000 total bits = 77.7% efficient

Synchronous transmission



Characteristics:

- Data is sent as a block of uninterrupted bytes
- Synchronization bytes precede and follow the data block
- Synchronization is maintained whether data is actually being sent and detected or not
- Modems remain synchronized during idle time

Efficiency (1000 byte transmission)

Control / overhead bits: 48 total control bits per block using HDLC
 48 control bits per block x 1 block = 48 control bits
 7000 data bits / 7048 total bits = 99.3% efficient

Figure 2-7 Asynchronous vs. Synchronous Transmission

A second key difference between synchronous and asynchronous transmission is what happens when the line is idle with no data being transmitted. In asynchronous communications all data transmission stops and the communicating devices fall out of synchronization. In synchronous communication the communicating devices continue to transmit special null characters and thus stay in synchronization. The difference between synchronous and asynchronous communications is further illustrated in Figure 2-7.

Half vs. Full Duplex Communication

Data communications sessions are bidirectional in nature. Even if the objective of the communication is to send a file from the sender to the destination some communication must go from the destination back to the sender. This reverse direction data includes error detection and correction information (covered later in this chapter). There are two environments available for handling this bidirectional traffic: full and half duplex.

In a full duplex communications environment both devices can transmit at the same time. This is analogous to an in-person conversation: The communications environment allows you to both listen and talk at the same time. In a half duplex environment you can only hear *or* talk at any given point in time. This is similar to a conversation taking place via a walkie-talkie: You can only listen until you push the talk button to transmit. Once you press the talk button you can only transmit until you release it to begin listening again. Similarly, when two data communication devices are communicating in a half duplex environment, one device must be the transmitter and the other the receiver. In order to enable bidirectional communication they periodically have to reverse roles.

This role reversal is known as **turnaround time** and—depending on the environment—can take two-tenths of a second or longer. This might not seem like a very long time, but if this role-reversal needed to be done several thousand times over the course of a communications session, it can significantly affect efficiency and by extension cost. Most communication technologies support full duplex communication in some manner, although it is often a configurable choice. Given the choice of full or half duplex, it is usually better to choose full duplex unless there is a significant penalty in transmission speed associated with full duplex.

Modulation/Demodulation

Although all modern data communications systems transmit digital data, depending on the type of communication channel used for the transmission it might have to be converted to an analog signal for transmission. In this case, the communication equipment at the sending end must convert the signal from its native digital format to an analog signal and transmit it to the destination. The communication equipment at the destination then takes the incoming analog signal, converts it back into a digital signal and relays it to the destination device. The process of converting digital data into an analog signal is known as modulation. Conversely, the process of converting a modulated analog signal back into a digital format is known as demodulation. These processes are illustrated in Figure 2-8.

Communications equipment that handles this conversion of data between digital and analog formats is known as a MODulator/DEMODulator, or **modem**. As shown in Figure 2-9, modems are placed at each end of the analog communication channel between the sending device and the channel.

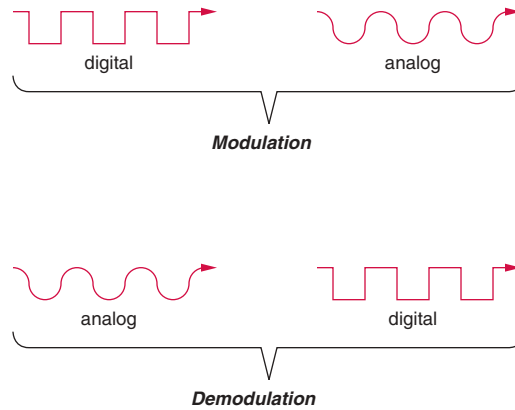


Figure 2-8 Modulation vs. Demodulation

Carrier Waves An analog communication channel is used to transmit data in the form of an analog wave. In order to represent the discrete-state ones and zeroes, or bits of digitized data on an analog communication channel it must be converted into different types of waves. A “normal” or “neutral” wave must exist to start with, that can be changed to represent these ones and zeroes.

This “normal” or “neutral” wave is called a **carrier wave**. A sample carrier wave, along with its characteristics, is illustrated in Figure 2-10. Modems generate carrier waves that are altered (modulated) to represent bits of data as ones and zeroes. Before they can communicate, two modems must establish a carrier wave between themselves. Once the carrier wave has been established, the actual data transmission can begin as the modems manipulate the carrier wave to represent ones and zeroes.

How can the carrier wave be manipulated to represent ones and zeroes? There are three physical characteristics of a wave that can be altered or modulated:

- **Amplitude**
- **Frequency**
- **Phase**

As will be seen later in this section, in some modulation schemes more than one of these characteristics can be modulated simultaneously.

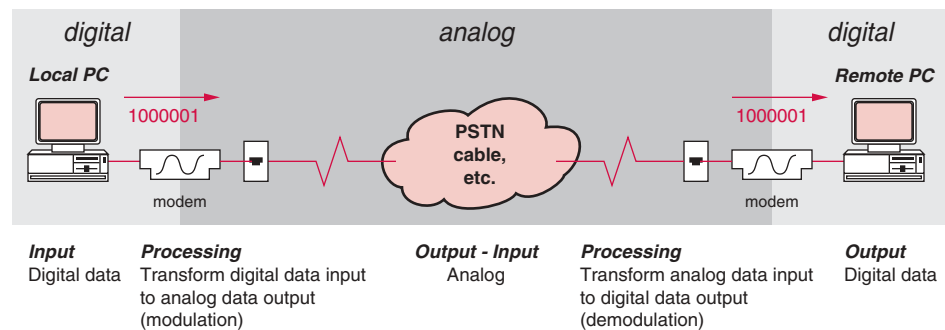


Figure 2-9 I-P-O Analysis: Modem Based Communication Channels

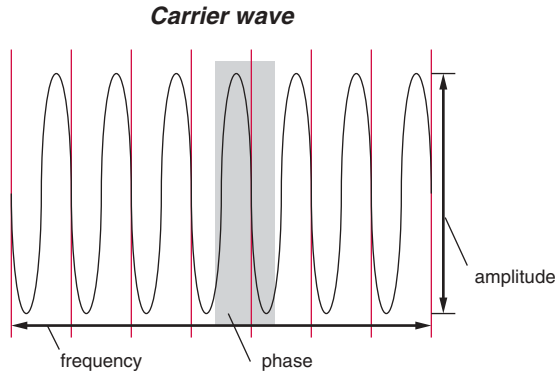


Figure 2-10 Carrier Wave

Amplitude Modulation Figure 2-11 illustrates **Amplitude Modulation** of a carrier wave. Only the amplitude changes; the frequency and phase remain constant. In this example, the portions of the wave with increased height (altered amplitude) represent 1's and the unaltered lower wave amplitude represent 0's. Together, this portion of the wave would represent the letter "A" using the ASCII-7 character encoding scheme.

Each of the vertical lines in Figure 2-11 separates opportunities to identify a 1 or 0 from another. These timed opportunities to identify ones and zeroes by sampling the carrier wave are known as signaling events. The proper name for one signaling event is a **baud**.

Frequency Modulation Figure 2-12 represents **frequency modulation** of a carrier wave. Frequency modulation is often referred to as **frequency shift keying**, or **FSK**. The frequency can be thought of as how frequently the same spot on two subsequent waves pass a given point. Waves with a higher frequency will take less time to pass while waves with a lower frequency will take a greater time to pass.

The distance between the same spots on two subsequent waves is called the **wavelength**; the longer the wavelength, the lower the frequency and the shorter the wavelength, the greater the frequency. In, Figure 2-12 the higher frequency (shorter wavelength) part of the wave represents a 1 and the lower frequency

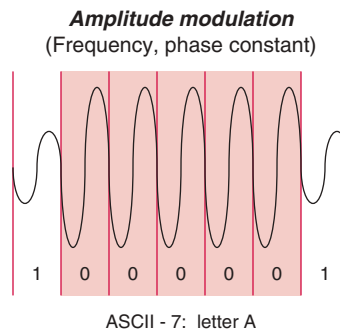


Figure 2-11 Amplitude Modulation

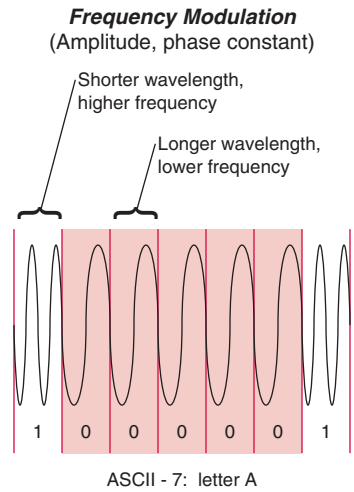


Figure 2-12 Frequency Modulation

(longer wavelength) part of the wave represents a 0, while amplitude and phase remain constant. Again, the entire bit stream represents the letter A in ASCII-7.

Phase Modulation Figure 2-13 illustrates an example of **phase modulation**, also known as **phase shift keying** or **PSK**. Although the phase varies, the frequency and amplitude remain constant. Phase modulation can be thought of as initiating a shift or departure from the “normal” continuous pattern of the wave. In Figure 2-13 the normal carrier wave would follow the broken line, but instead the phase suddenly shifts and heads off in another direction. This phase shift of 180 degrees is a detectable event with each change in phase representing a change in state from 0 to 1 or 1 to 0 in this example.

Measuring Phase Shift In Figure 2-13, the detected analog wave was either the base carrier wave with no phase shift, or it was phase shifted 180 degrees. Given that phase shifts are measured in degrees, there are multiple amounts of phase shift that can be applied to a carrier wave. By increasing the number of possible phase shifts, the number of potential detectable events can be increased. As illustrated in Figure 2-14, when there are just two potential detectable events (no phase shift, or 180 degree

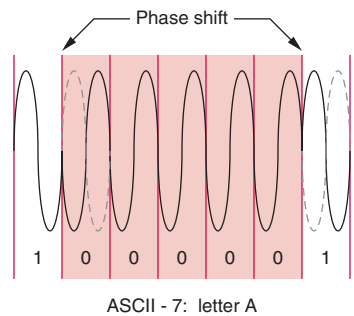


Figure 2-13 Phase Modulation

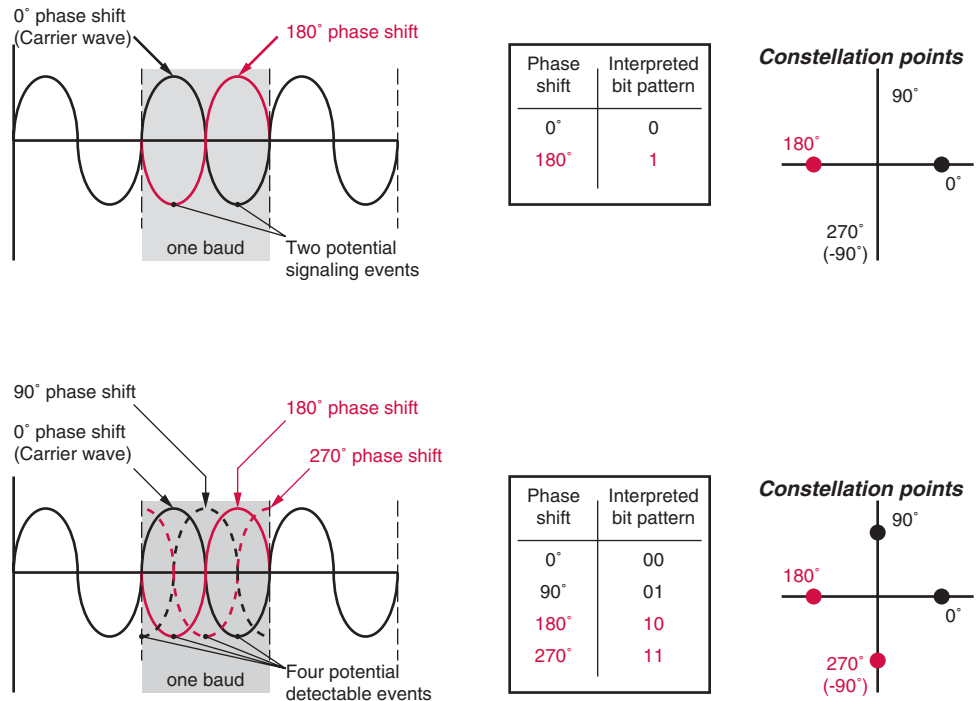


Figure 2-14 Relationship between number of phase shifts and number of potential detectable events

phase shift), the two events represent either a 1 respectively. However, by introducing four potential phase shifts (0, 90, 180, 270), 2 bits can be associated with each detectable event.

A simpler and perhaps clearer way to represent phase shifts as illustrated in Figure 2-14 is through the use of **constellation points**. Using a four-quadrant representation of the 360 degrees of possible phase shift, individual points represent each different shifted wave. Note that when represented in a constellation diagram, a phase shift of 270 degrees is represented as -90 degrees. Phase shift modulation with four different phases is more properly referred to as **quadrature phase shift keying** or **QPSK**.

Baud Rate vs. Transmission Rate The number of signaling events per second is more properly known as the **baud rate**. Although baud rate and **bps** (bits per second) or **transmission rate** are often used interchangeably, the two terms are in fact related, but not identical. In the first illustration in Figure 2-14, only two detectable events were possible meaning that only one bit was interpreted at each signaling event (one bit/ baud). Therefore, in this case the baud rate was equal to the transmission rate as expressed in bps (bits per second).

However, in the second illustration in Figure 2-14, four detectable events are possible for each signaling event, making it possible to interpret two bits per baud. In this case, the bit rate or transmission rate as measured in bps would be twice the baud rate.

More sophisticated modulation techniques are able to interpret more than one bit per baud. In these cases, the bps is greater than the baud rate. For example, if the baud rate of a modem was 2,400 signaling events per second and the modem was able to interpret 2 bits per signaling event, then the transmission speed would be

4,800 bps. Mathematically, the relationship between baud rate and transmission rate can be expressed as follows:

$$\text{Transmission rate (bps)} = \text{Baud rate} \times \text{bits/ baud}$$



MORE THAN ONE BIT/BAUD

There are really only two ways in which a given modem can transmit data faster:

1. As mentioned previously, increase the signaling events per second, or baud rate.
2. Find a way for the modem to interpret more than one bit per baud.

By modifying a phase modulation technique such as that illustrated in Figure 2-14, a modem can detect, interpret, and transmit more than one bit per baud. The mathematical equation that describes the relationship between the number of potential detectable events and the numbers of bits per baud that can be interpreted is as follows:

$$\text{Number of states} = \text{Number of potential detectable events}^{\text{bits/ baud}}$$

- Number of states = always 2 (Data are either a 1 or 0)
- Number of potential detectable events = 4 different phase angles (0, 90, 180, 270) as illustrated in the second illustration in Figure 2-14.

To solve:

- To what power must 2 (the number of states) be raised to equal 4 (the number of different detectable events)?

The answer is 2, meaning that 2 bits/ baud can be interpreted at a time. Two bits at a time are known as a dibit. By extending the mathematical equation above, it should be obvious that:

Number of Potential Detectable Events	Number of bits/ baud	Also known as
8	3	tribit
16	4	quadbit
32	5	
64	6	
128	7	
256	8	
512	9	

Figure 2-15 shows the encoding of an ASCII A using dibits in differential Quadrature Phase Shift Keying. In this differential approach the phase change is measured in relationship to the previous baud rather than from the original carrier phase. Before

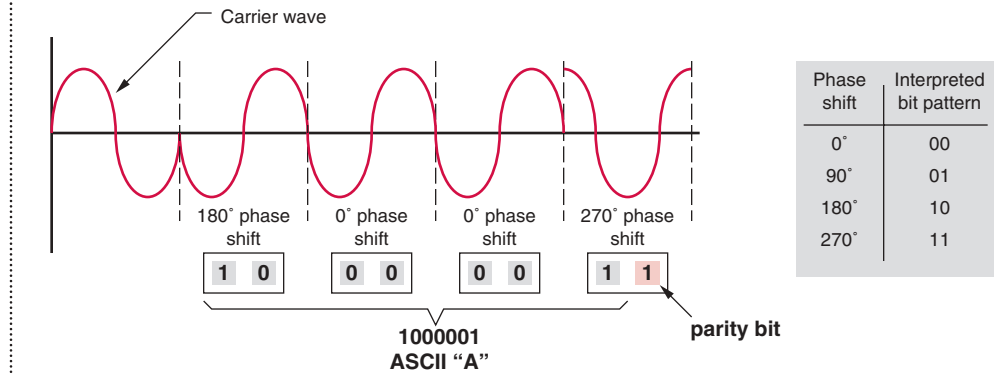


Figure 2-15 Differential Quadrature Phase Shift Keying

the ASCII A can be encoded, an eighth bit must be appended to make the total number of bits divisible by 2. This eighth bit is known as a parity bit and is further explained later in the chapter.

Comparing Figure 2-15 with Figure 2-13 shows the increased efficiency of QPSK over simple PSK. Using QPSK, only four baud periods are required to transmit the ASCII “A” using dibits, compared to the seven baud periods required for two-state phase modulation. Given a constant frequency, the data can be transmitted almost twice as fast.

Quadrature Amplitude Modulation As illustrated in the previous example, increasing the number of phase shifts increases the efficiency (and, by extension, the transmission rate) of the communication channel. Depending on the type and quality of the analog transmission line, it may be possible to further increase the number of phase shifts that can be detected. If the communication channel was good enough to support 16 different measurable phase shifts (phase shifts of as little as 22.5 degrees) the effective speed of the line could once again be doubled.

However, there is a method of further increasing the efficiency of the communication channel even if it does not support the reliable detection of more than four phase shifts. By combining the modulation of two aspects of the carrier wave, it is possible to add additional signaling events. When modulating multiple aspects of a carrier wave it is most common to modulate the phase and amplitude, while leaving the frequency steady. By varying both phase and amplitude 16 different detectable events can also be produced. This technique is known as **quadrature amplitude modulation** or **QAM**.

16QAM, with 16 different potential detectable events, allows 4 bits/baud or quadsbits to be produced or detected per signaling event, resulting in a transmission rate four times the baud rate. Figure 2-16 illustrates a representative set of constellation points and associated quadsbits for a QAM modulation scheme. Differences in phase are represented in degrees around the center of the diagram, while differences in amplitude are represented by linear distance from the center of the diagram. Each point is uniquely identified by combining 1 of 3 potential amplitudes (.311V, .850V and 1.161V) with 1 of 12 potential phase shifts (15, 45, 75, 105, 135, 175, -165, -135, -105, -75, -45, -15 degrees). Obviously, all potential combinations of these two sets of variables are not used in 16QAM.

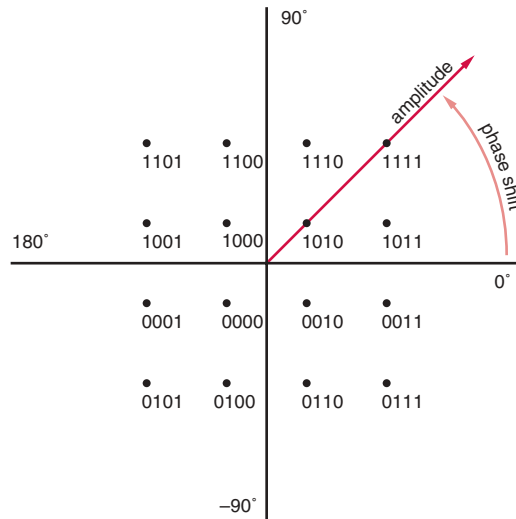


Figure 2-16 QAM Constellation Points and Quadbits



In Sharper Focus

NYQUIST'S THEOREM AND SHANNON'S LAW

What are the underlying factors that limit the carrying capacity of a given circuit? The work of Harry Nyquist and Claude Shannon helps to answer that question.

Nyquist's Theorem The constellation points illustrated in Figure 2-16 are sometimes referred to as symbols. As the number of constellation points (symbols) increases and symbols are in closer proximity to each other on the constellation diagram, the chance for a modem to misinterpret constellation points increases. Interference between symbols that can cause misinterpretation is known as **intersymbol interference**. Nyquist investigated the maximum data rate (measured in bps) that can be supported by a given bandwidth (measured in Hz) due to the effect of intersymbol interference. He found the relationship between bandwidth (W) and maximum data rate (C) to be

$$C = 2W$$

However, this fails to account for the ability of modern modems to interpret more than 1 bit per baud by being able to distinguish between more than just two symbols or potential detectable events. Taking this ability into account with the number of potential detectable events represented as M , Nyquist's Theorem becomes

$$C = 2W \log_2 M$$

A practical example of Nyquist's theorem is that of a modem that uses dial-up telephone lines. The bandwidth of an analog voice transmission channel is 3,100 Hz. Modern dial-up modems allow for 16 detectable events per baud using QAM. Therefore, Nyquist states that maximum throughput of a voice-grade analog communications channel is $2(3,100) \times \log_2(16)$, or 24,800 bps.

Shannon's Law The data rates theorized by Nyquist's Theorem are often not achievable in practice due to the presence of noise in the analog communication channel.

Noise is measured as a ratio of the strength of the data signal to the strength of the background noise. This ratio is known as the **signal to noise ratio (S/N)** and is computed as follows:

$$(S/N)_{dB} = 10 \log (\text{signal power}/\text{noise power})$$

S/N is expressed in decibels ($_{dB}$). Decibels are a logarithmic measurement using a reference of 0_{dB} for comparison. As a result, a noise level of 10_{dB} is 10 times more intense than a noise level of 0_{dB} , a noise level of 20_{dB} is 100 times more intense than a noise level of 0_{dB} , and a noise level of 30_{dB} is 1,000 times more intense than a noise level of 0_{dB} .

Shannon found that the higher the data rate, the more interference is caused by a given amount of noise, thus causing a higher error rate. This should make sense because at higher data rates, more bits are traveling over a circuit in a fixed length of time, and a burst of noise for the same length of time will affect more bits at higher data rates. By taking into account the signal-to-noise ratio, Shannon expresses the maximum data rate of a circuit (C) as

$$C = W \log_2 (1 + S/N)$$

To make calculating Shannon's Law easier for calculators that aren't capable of calculating a \log_2 , this equation is mathematically equivalent to

$$C = 3.32 W \log_{10} (1 + S/N)$$

Considering a voice-grade circuit: if $W = 3,100$ Hz and the $S/N = 30_{dB}$, then $C = 30,894$ bps. Depending on the value inserted into Shannon's Law for S/N or for W , C will vary accordingly. 24,000 bps is also a common value for C using Shannon's Law. As illustrated by substituting different values for W and for S/N , the data channel capacity (C) can be more drastically affected by changes in bandwidth (W), than by changes in the signal to noise ratio (S/N).

Finally, it should be noted that there are numerous other line impairments such as attenuation, delay distortion, and impulse noise that Shannon's Law does not take into account. As a result, the data channel capacity (C) derived from Shannon's Law is theoretical and is sometimes referred to as error-free capacity.

Data Compression

In the world of data communication, speed is expensive. The faster the data need to be sent, the more the equipment and services are required to send it cost. One way to get additional data throughput is to employ data compression. Data compression involves the sending device replacing large strings of repeating character patterns with a special code that represents the pattern. The code is then sent to the destination device. From that point forward, the sending device replaces any instances of the original pattern with the code. As the code is significantly smaller than the pattern it represents, the amount of data sent between the two devices is reduced as compared to sending the raw data. This reduction, up to 400 percent under optimal conditions, results in an increased amount of data being sent between the sending device and the receiving device (also known as throughput). Figure 2-17 illustrates the difference between throughput and transmission rate.

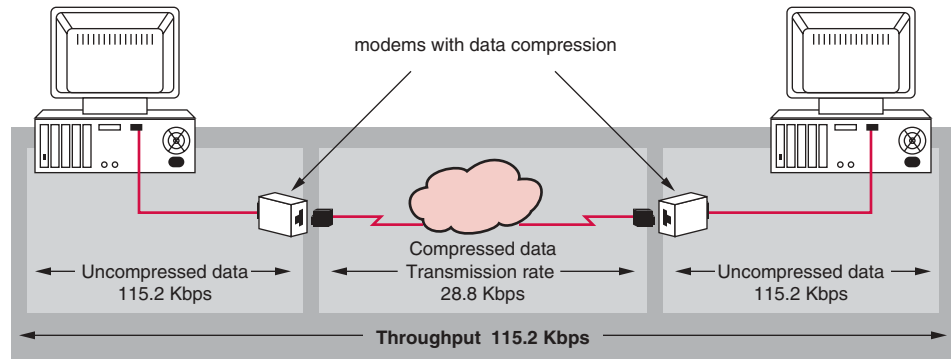


Figure 2-17 Data Compression: Transmission Rate vs. Throughput

Given the transmission of compressed data in which a single character may “uncompress” into many characters, the importance of error prevention, detection, and correction when using data compression cannot be overstated. An erroneously transmitted character could be referenced in the receiving device’s library of repetitive patterns and incorrectly “uncompressed” into the wrong data stream.

It is important to understand how data compression works and the fact that all of the data does not actually travel across the communication channel. It is equally important to understand that data compression only works if both the sending and receiving data communication devices support the same data compression standard. Data compressed by the sending device must be uncompressed by the receiving device using an identical algorithm or methodology.

Obviously, the more “repetitive patterns” the data compression algorithm can find in the data, the higher the compression ratio that can be expected. Thus, some data streams are more compressible than other data streams and therefore yield a higher data compression ratio. It is also important to note that all data compression technologies use a similar process. If the raw data stream has already been compressed through the use of a CODEC such as an MP3 file or a compression utility such as a zip file, no gain will be realized by the data communication device’s compression algorithm. In fact, in such an environment it is possible that the amount of data transmitted could slight increase if data compression is enabled.



Practical Advice
and Information

SOFTWARE VS. HARDWARE DATA COMPRESSION

The data compression techniques discussed so far are hardware based, implemented in the communication devices themselves. Many data-communication software solutions (such as Microsoft Windows Dial-up networking and many virtual private network solutions) also include data compression. If both software and hardware compression are available, software compression is usually more effective because the software packages are typically optimized for the type of data being sent. Hardware compression must be generic as the device designers have no idea of the type of data they might be transmitted. Regardless of whether hardware or software compression is chosen, both hardware and software compression should never both be enabled at the same time.

Principle of Shifting Bottlenecks The ability to compress data introduces another key data communications issue: the **Principle of Shifting Bottlenecks**. Just as a chain is only as strong as its weakest link, a data communications system is only as fast as its

slowest link—its bottleneck. If the existing bottleneck is eliminated (by employing data compression or any other technique), then the next slowest link becomes the bottleneck. Before investing in resolving one bottleneck, it should be determined what the next bottleneck would be. If investing in a solution to remove the first bottleneck will only result in a small increase in overall throughput before the next bottleneck becomes the limit, it might not make economic sense to do anything.

■ DATA COMMUNICATION TECHNIQUES

Packetization

Packetization is the process of dividing the data stream flowing between devices into structured blocks known as packets. A **packet** can be defined as a group of bits organized in a predetermined, structured manner consisting of a piece of the overall data stream to which overhead or management information is added to assure error-free transmission of the data to its destination. These packets may be referred to as messages, sessions, frames, cells, blocks, data units, or several other names, depending on which layer of the OSI Network Reference Model they are located. The packetization process is illustrated in Figure 2-18.

The predetermined, structured nature of a packet is critical. At the lowest level the packet itself consists of nothing more than a series of ones and zeroes; there is nothing in the packet itself that explicitly indicates what any of the bits mean. Rather, the bits have implicit meaning; through the use of standards the two communicating devices know the number of bits assigned to the fields in the header, data section, and trailer of the packet. Fields used in the header to enable the seamless communication of data include destination and source addresses, sequence numbers, and error checks.

Encapsulation/De-encapsulation

Once the data is packetized, it is placed in protocol containers for transmission across the network. As illustrated in Figure 2-19, a data message emerges from an application

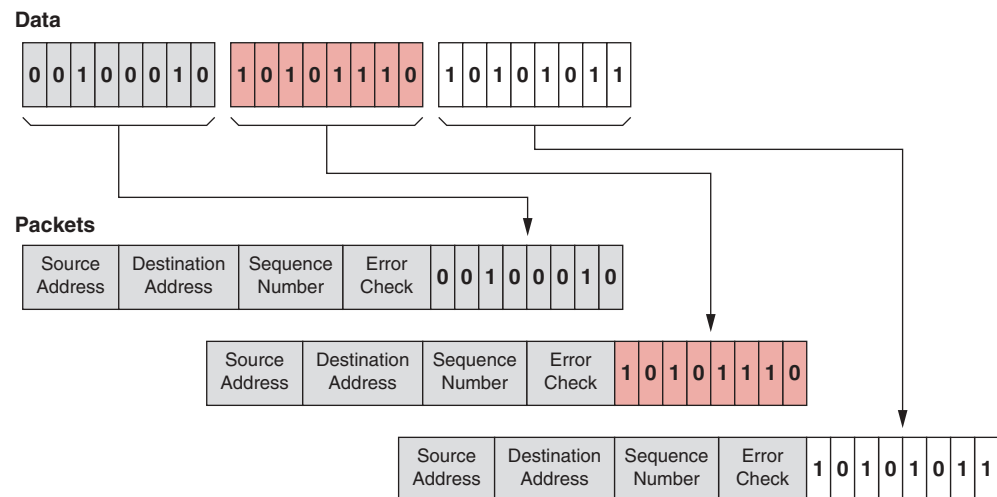


Figure 2-18 Packetization Process

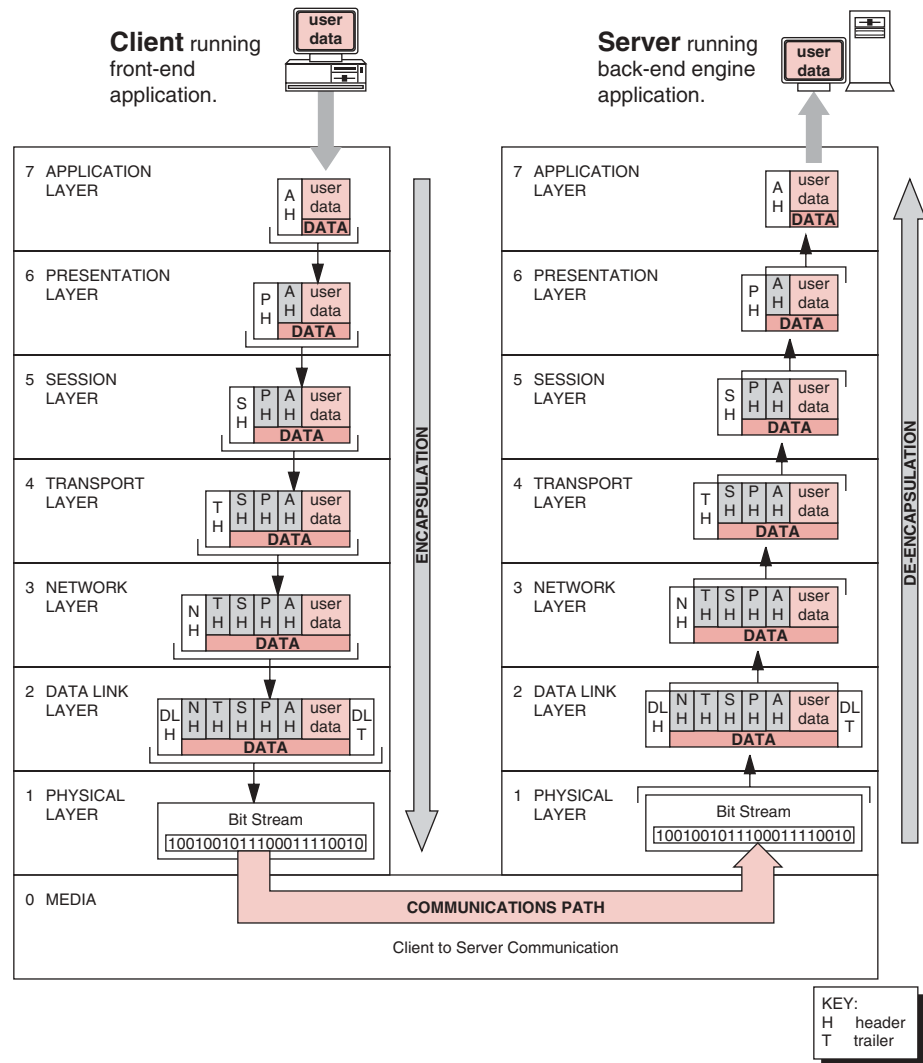


Figure 2-19 Encapsulation and De-encapsulation in the OSI Network Reference Model

and proceeds down the protocol stack of the network operating system in a process known as **encapsulation**. Each successive layer of the OSI model adds a header according to the syntax of the protocol that occupies that layer. In the case of the data-link layer, both a header and trailer are added. The bit stream is finally passed along the media that connects the two computing devices. Although the OSI model may seem to imply that given layers in a protocol stack talk directly to each other on different computers, the fact is that the computers are only physically connected by the media and that is the only layer where there is a direct communications link between computers.

When the full bit stream arrives at the destination computer, the reverse process of encapsulation, known as **de-encapsulation**, takes place. Each successive layer of the OSI model removes headers and/or trailers and processes the data that was passed to it from the corresponding layer protocol on the source computer until the data is given to the destination application running on the destination computer. While Figure 2-19 focused on the OSI network reference model, the same concepts are utilized in all other protocol models including the TCP/IP model.

Multiplexing

Multiplexing is the process of sharing the bandwidth of a data line among multiple communication sessions. As illustrated in Figure 2-20, a long-distance parcel shipping analogy can be used to illustrate the underlying technical concept of multiplexing.

Multiplexing refers to taking a single communication channel and breaking it into subchannels that can be used to carry independent messages. This is shown in Figure 2-20; after the data is packetized, the packets are multiplexed into a single truck for transmission. In data communications, a multiplexer (also known as a **MUX**) takes data packets and packages them for transmission over a shared connection along with data packages from other sources. Once the package is received at the destination address, the data packets are de-multiplexed and forwarded to their respective destinations. These destinations represent the applications and services that require connectivity. Examples include web browsing and instant messaging.

Three basic techniques are employed in multiplexing digitized traffic across electrical physical connections where bits of data are transmitted as discrete voltages of electricity:

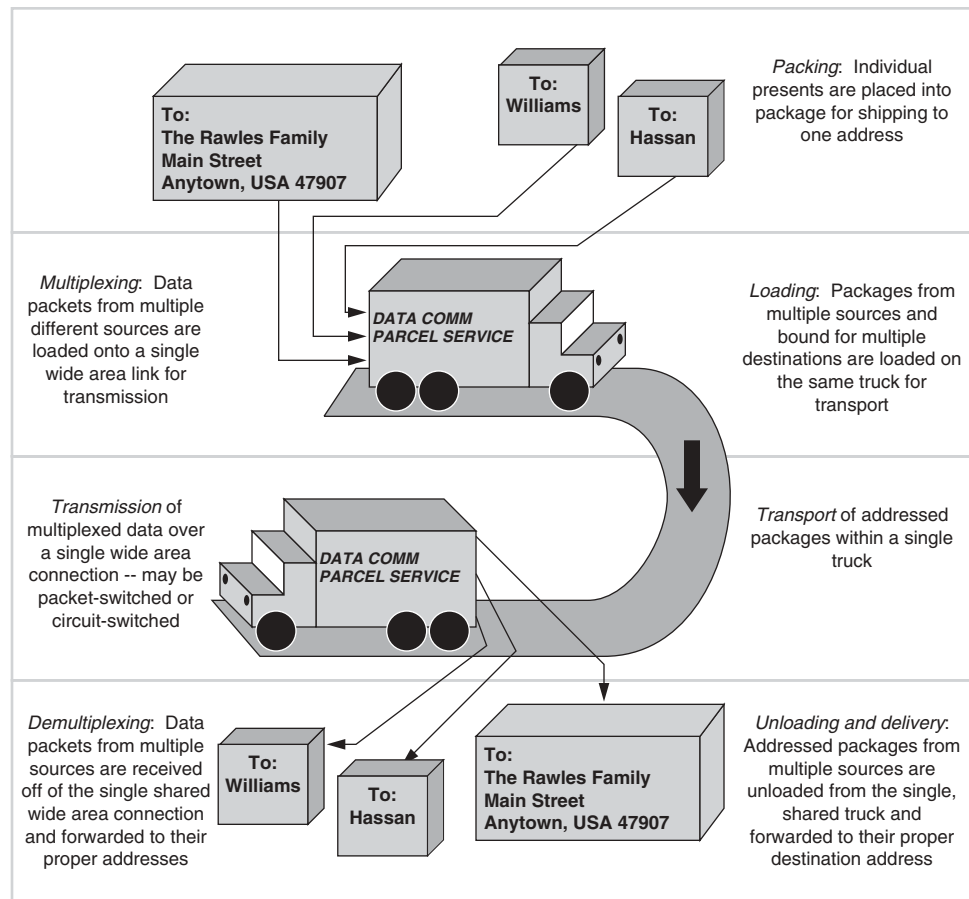


Figure 2-20 Multiplexing: A Parcel Shipping Analogy

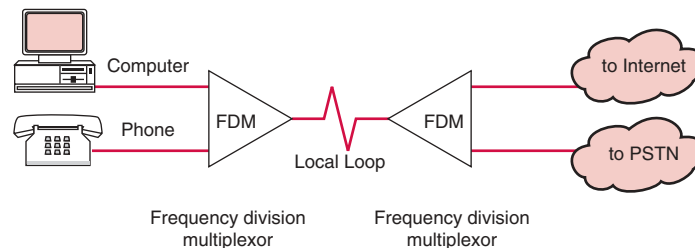
- **Frequency Division Multiplexing (FDM)**
- **Time Division Multiplexing (TDM)**
- **Statistical Time Division Multiplexing (STDM)**

When optical transmission is used, bits of data are represented by bursts of light energy of varying wavelengths. In this environment, a relatively new multiplexing technique known as **wavelength division multiplexing (WDM)** has been developed. WDM technology is deployed primarily by telecommunications carriers with extensive long distance fiber optic networks in order to increase transmission capacity up to tenfold without the need to install additional fiber. WDM will be discussed in more detail in chapter 7.

Frequency Division Multiplexing In frequency division multiplexing, multiple input signals are modulated to different frequencies within the available bandwidth of a single circuit and subsequently demodulated back into individual signals on the output end of the composite circuit. Sufficient space in between these separate frequency channels is reserved in **guardbands** in order to prevent interference between the two or more input signals that are sharing the single circuit. Figure 2-21 illustrates a simple frequency division multiplexing configuration.

The device that implements frequency division multiplexing is known as a frequency division multiplexer or FDM. An FDM can be either a stand-alone unit or it can be integrated into another piece of data communications equipment such as a cable or DSL modem.

Overall Configuration



Inside the Local Loop

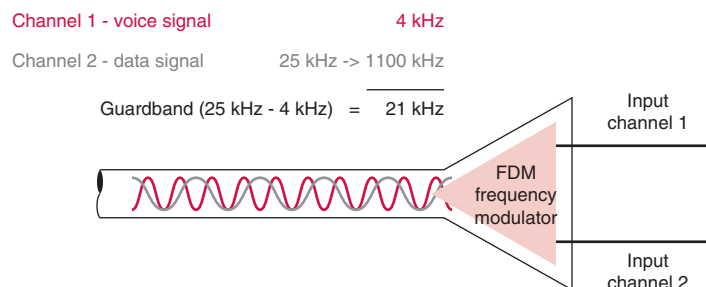


Figure 2-21 Frequency Division Multiplexing

In FDM, the total bandwidth of the composite channel is divided into subchannels by frequency. FDM works in the same manner as a cable television system; each channel is given a specific amount of bandwidth on the cable. By changing the frequency the television samples different channels are tuned for viewing.

In an FDM implementation, each channel is available 100 percent of the time, just as in a dedicated circuit. As a result, the timing of signals between a connected terminal or device and the centralized processor are not affected. However, the amount of bandwidth available in each channel is limited to the bandwidth devoted to the channel. If one channel is not using all of its available bandwidth there is no way to allow it to be used by another channel.

Common uses of FDM include modern Internet access technologies such as DSL (digital subscriber line) and cable modem services. In a DSL solution, FDM is used to multiplex a high-speed data signal over the same physical pair of copper wires used for dial-up telephone calls. In a cable modem solution, the high-speed data signal is multiplexed onto the same physical coaxial cable used to deliver television programming from the cable company. More detail on DSL and cable modems is presented in chapter 3.

Time Division Multiplexing In a sense, time division multiplexing (TDM) works just the opposite of FDM. Rather than breaking the bandwidth into channels of lesser capacity that are available at all times, TDM creates channels by allowing the data communications devices to use all of the available bandwidth for short periods of time. This is similar to playing a game of “hot potato”; when you have the potato you can use all of the bandwidth but you have to wait to transmit until you have the potato. The portion of time available to each connected input device (known as the **time slot**) is constant and controlled by the Time Division Multiplexer. Unlike FDM where a percentage of the available bandwidth is lost to guardbands, in a TDM environment all of the bandwidth is available for use. However, because each device can only transmit during its timeslot the timing of communications between the devices becomes more complicated.

Another key point to understand about time slots in a TDM environment is that a fixed portion of time, measured in milliseconds, is reserved for each attached input device whether the device is active or not. As a result, efficiency is sacrificed for the sake of simplicity. Just as in FDM, if a channel (time slot) is not being fully utilized there is no way to give that bandwidth to another channel that is using all of its bandwidth; a device with nothing to transmit is given its full time allotment while other busy devices must wait their turn to transmit. Figure 2-22 illustrates simple time division multiplexing.

As can be seen in Figure 2-22, each input channel has a fixed amount of buffer memory into which it can load data. Flow control is used to tell each device to stop transmitting to the buffer when it fills. A **central clock** or timing device in the TDM gives each device its allotted time to empty its buffer into an area of the TDM where the combined data from all of the polled input devices is conglomerated into a single message frame for transmission over the composite circuit. This process of checking on each connected terminal in order to see if any data is ready to be sent is known as **polling**.

If a device is inactive with nothing in its input buffer to contribute to the consolidated message frame, its allotted space in the **composite message frame** is filled with blanks. The insertion of blanks, or null characters, into composite message links is the basis of TDM's inefficient use of the shared composite circuit connecting the

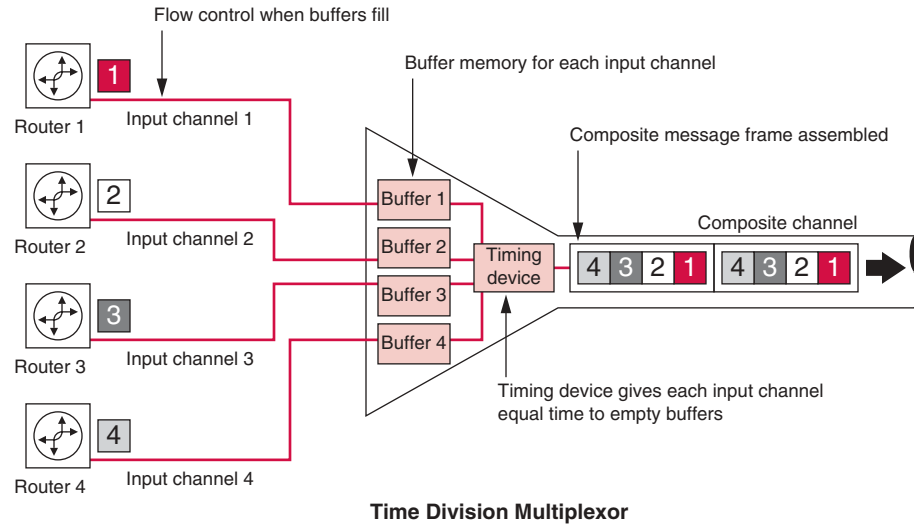


Figure 2-22 Time Division Multiplexing

two TDMs. Although the data from the devices is combined into a single message frame, each individual device's data is still identifiable by its position in the composite message frame. This is important since once the composite message frame has finished its journey across the transmission link, it must be re-segmented back into the individual device data streams.

Statistical Time Division Multiplexing Statistical time division multiplexing (STDM) is a variant of FDM that seeks to offer more efficient use of the composite bandwidth than simple TDM by employing increased monitoring and manipulation of input devices to accomplish two major goals:

- Eliminate “idle time” allocations to inactive terminals.
- Eliminate padded blanks or null characters in the composite message blocks.

In STDM, time slots are dynamically allocated to input devices rather than being fixed. As devices become more active, they get more time slots to use. As devices become less active, the STDM MUX polls them less frequently. This dynamic time slot allocation takes both processing power and additional memory. Statistics are kept as to terminal activity over time and hence the name: *statistical time division multiplexing*. Specially programmed microprocessors and additional buffer memory are key components of STDMS and contribute to their increased costs over the simpler TDMs.

To increase the efficiency of use of the composite link, padded blanks and null characters are not inserted into message frames for inactive terminals. In an STDM, rather than assign space to input devices in the composite message frame by position regardless of activity, the STDM adds control information to each terminal's data within the composite message frame that indicates the source device and how many bytes of data came from that device. Figure 2-23 illustrates composite message block construction in STDMS.

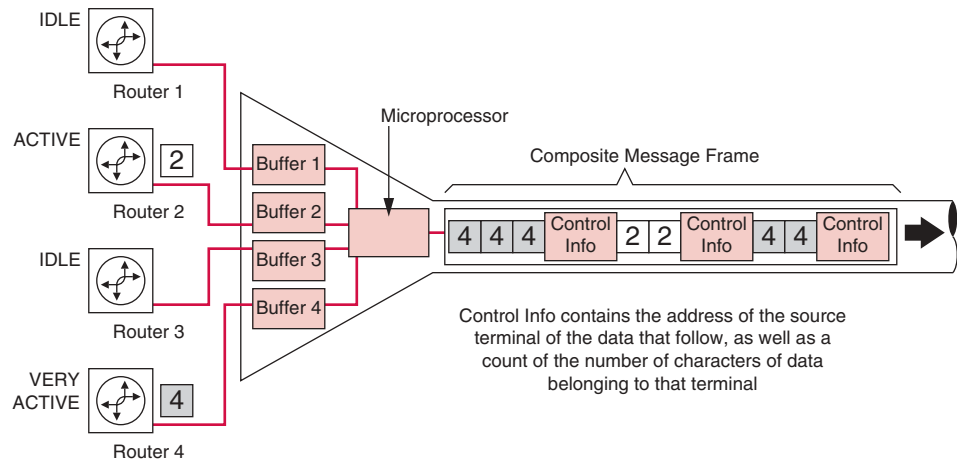


Figure 2-23 STDMs Make Efficient Use of Composite Bandwidth



Practical Advice
and Information

NETWORK TIMING CONSIDERATIONS

The dynamic allocation of time afforded to individual end-devices by the STDM can interfere with the timing between these devices. It is particularly important to utilize a common synchronous *building integrated timing source* (BITS) clock to maintain synchronization across the entire network.

Switching

If a device wishes to communicate with another device to which it has a direct connection, it is relatively simple to establish a connection and begin transmitting data. In this case the local device knows that any data sent out the specified interface will be travel directly to the destination across the direct connection, or circuit, between the two devices. However, when a device wishes to communicate with another device to which it does not have a direct connection, the process becomes significantly more difficult. Before the local device can begin sending data to the destination, a connection must be established to the destination. As shown in Figure 2-24, this connection must go through intermediate devices that create a path to the destination. Depending on the technology and protocols used, these intermediate devices may be known as exchanges, bridges, LAN switches, or routers.

Switching allows temporary connections to be established, maintained and terminated between sources and destinations, sometimes referred to as **data sinks**. There are two primary switching techniques employed in switching architectures:

- Circuit switching
- Packet switching

In circuit-switched networks, users get dedicated bandwidth on circuits created solely for their use. In packet-switched networks, users' packetized data is transported across circuits between packet switches, along with the data of other users of the same packet switched network.

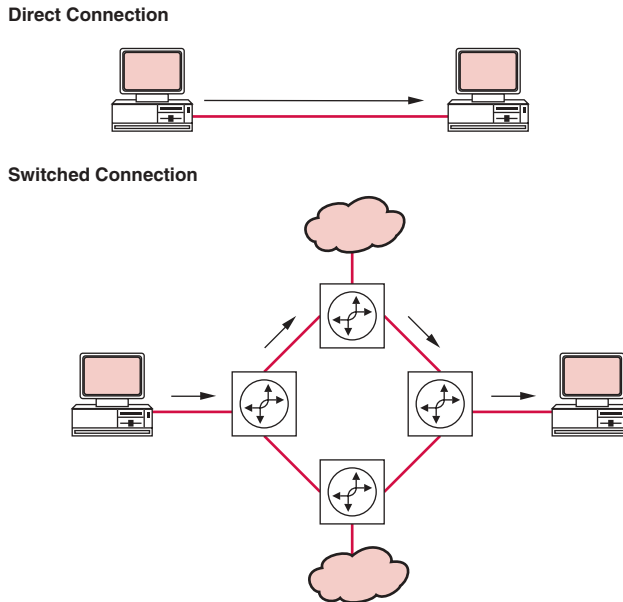
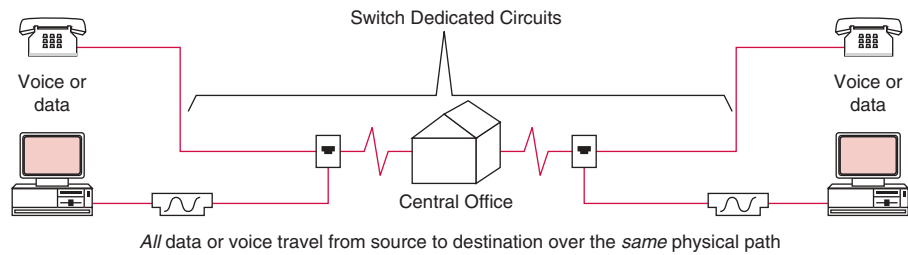
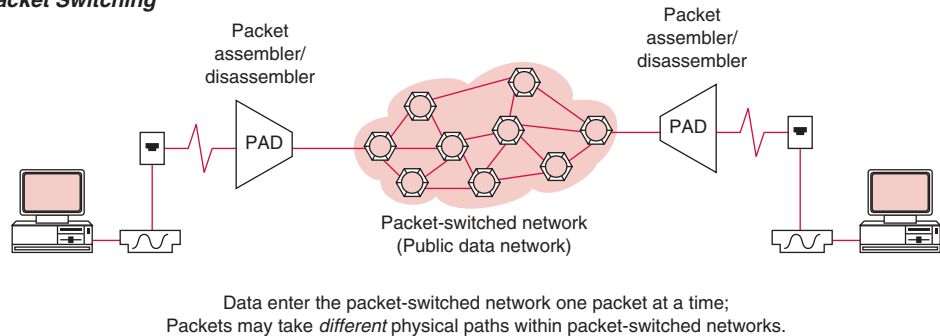


Figure 2-24 The Need for Switching

Circuit Switching In a **circuit-switched network**, the work to create a signal path is done up front; a switch fabric creates a direct electrical or optical path between the source and the destination. After the circuit is established, communication takes place just as if the temporary circuit were a permanent direct connection: The switched dedicated circuit established on circuit-switched networks makes it appear to the user of the circuit as if a wire has been run directly between the communicating devices. The physical resources required to create this temporary connection are dedicated to that particular circuit for the duration of the connection. If system usage should increase to the point where insufficient resources are available to create additional connections, additional requests for connection are denied.

Traditionally, the most common example of a circuit-switched network is the telephone system or Public Switched Telephone Network (PSTN). When making a call over the PSTN the user dials the address of the destination device to which they wish to communicate and the telephone switches create a circuit between the source and destination handsets. Signals leaving the source handset are directly sent across this dedicated circuit to the destination handset where they are reconstituted into the voice of the calling party.

Packet Switching In a **packet-switched network**, packets of data travel one at a time from the message source to the message destination. A packet switched network, otherwise known as a **public data network (PDN)**, is usually represented in network diagrams by a cloud symbol. Figure 2-25 illustrates such a symbol as well as the difference between circuit switching and packet switching. The cloud is an appropriate symbol for a packet-switched network because all that is known is that the packet of data goes in one side of the network and comes out the other. The physical path taken by one packet may be different than that taken by the other packets in the data stream and in any case, is unknown to the end user. Inside the

Circuit Switching**Packet Switching****Figure 2-25** Circuit Switching vs. Packet Switching

cloud is a series of **packet switches** that pass packets among themselves as they travel from source to destination.

Packets are specially structured groups of data that include control and address information in addition to the data itself. These packets must be assembled (control and address information added to data) somewhere before entry into the packet switched network and must be subsequently disassembled upon leaving the packet switched network before delivery to their destination. This packet assembly and disassembly is done by devices known as **PADs** or **packet assembler/disassemblers**. PADs may be standalone devices or may be integrated into other data communication devices such as modems or multiplexers. The PADs may be located directly at an end-user location, or may be located further along the communication path at the entry point to the packet-switched data network.

Another way in which packet switching differs from circuit switching is that as demand for transmission of data increases on a packet-switched network, additional users are not denied access to the packet-switched network. Overall performance of the network may suffer, errors and retransmission may occur, or packets of data may be lost, but all users experience the same degradation of service because in the case of a packet-switched network data travels through the network one packet at a time; traveling over any available path within the network rather than waiting for a switched dedicated path as in the case of the circuit-switched network.

Connectionless vs. Connection-Oriented Networks In order for any packet switch to process any packet of data bound for anywhere, it is essential that packet address information be included on each packet. Each packet switch then reads and processes the packet by making routing and forwarding decisions based on the packet's current location, destination address, and network conditions.

Because an overall data message is broken up into numerous pieces by the packet assembler, these message pieces may arrive out of order at the message destination if they take different paths across the packet switched network. The data message must be pieced back together in proper order by the destination PAD before final transmission to the destination address. These self-sufficient packets containing full source and destination address information, plus a sequence number, are known as **datagrams**. Figure 2-26 illustrates the packet assembly/disassembly process.

A **connectionless** packet network is a switching methodology in which each datagram is handled and routed to its ultimate destination on an individual basis, resulting in the possibility of packets traveling over a variety of physical paths on the way to their destination.

There are no error control techniques applied by a datagram-based or connectionless packet-switched network. Such a network requires the end devices (computers) to provide adequate error control. This lack of inherent error control is the reason that connectionless packet networks are also known as **unreliable** packet networks.

In contrast to unreliable connectionless packet networks, **connection-oriented**, or **reliable**, packet networks offer built-in error control provided by the packet network itself. This removes the requirement that the end user devices provide this functionality. Figure 2-27 contrasts connectionless and connection-oriented packet-switched networks.

■ ERROR CONTROL TECHNIQUES

The objective of every data communications session is to efficiently and accurately transmit the desired data. However, just as in communication between two people,

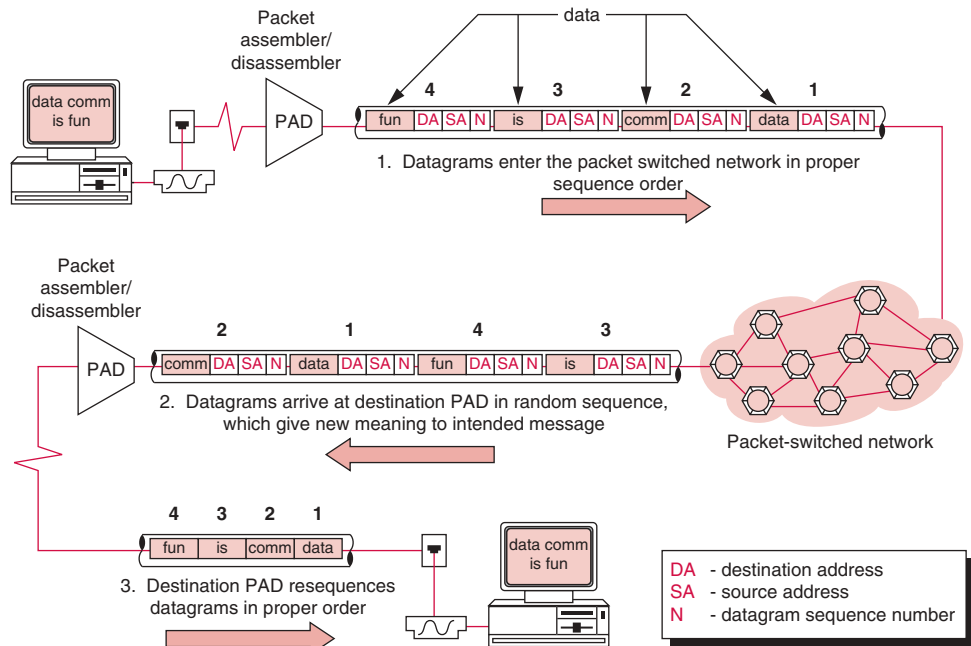


Figure 2-26 Datagram Delivery on a Packet Switched Network

	Overhead	Greatest Strength	Call Set-up	Addressing	Also Known As...	Error Correction	Flow Control
Connectionless	Less	Ability to dynamically reroute data	None	Global	Datagram unreliable	Left to end-user devices	Left to end-user devices
Connection-oriented	More	Reliability	Yes	Local logical channel number	Reliable Virtual circuit	By virtual circuit	By virtual circuit

Figure 2-27 Connection-Oriented vs. Connection Packet-Switched Networks

there are times when the message sent is not the message received due to inaccuracies in the transmission or reception equipment or anomalies in the communications channel. For a data communication session to be useful, there must be a means of handling any transmission problems to ensure the reliable transmission of accurate data.

Error Prevention

Data transmission errors occur when received data are misinterpreted due to noise or interference on the communication lines over which the data message traveled. Errors can be prevented by employing filters, amplifiers, and repeaters in the communication channel to improve the signal to noise ratio to reduce the likelihood of an error occurring. A detailed analysis of amplifiers and repeaters is provided in chapter 3.

Another way in which errors can be prevented during data transmission is through the use of **adaptive protocols**. Adaptive protocols adjust transmission session parameters such as base transmission speed in response to varying line conditions. The techniques available to help prevent errors vary, depending on the data transmission technologies being used. However, errors are inevitable; no amount of error prevention can eliminate them.

Error Detection

Once everything possible has been done to prevent errors, the focus shifts to reliably detecting the errors that do occur. Remembering that transmitted data, on the most elementary level, is merely a stream of ones and zeroes, the role of error detection can be defined as providing the assurance that the receiving computer receives the same ones and zeroes in the exact sequence that they are transmitted by the transmitting computer.

This assurance is achieved through the cooperative efforts of the transmitting and receiving data communication devices. In addition to transmitting the actual data, the transmitting device must also transmit some type of verifiable bit or character that the receiving device can use to decide whether the transmitted data was received correctly. Figure 2-28 illustrates this overall process shared by all error-detection techniques.

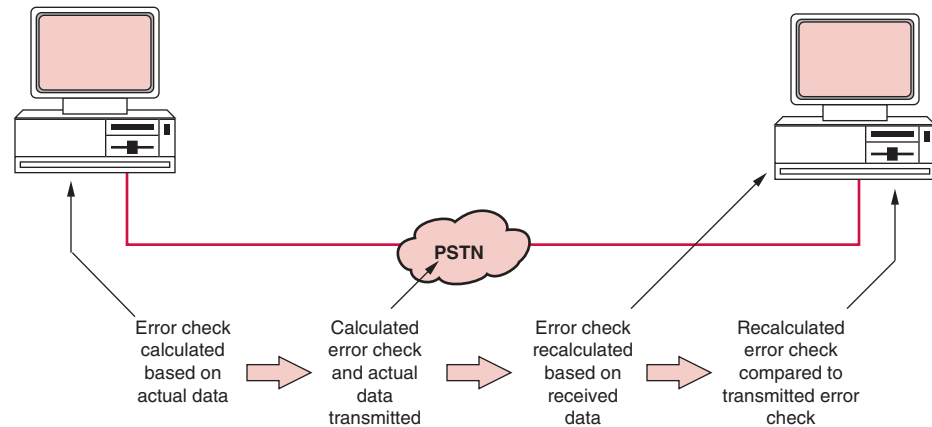


Figure 2-28 The Overall Error Detection Process

This generalized error detection process can be summarized as follows:

- The transmitting and receiving devices agree on how the error check is to be calculated.
- The transmitting device calculates and transmits the error check along with the transmitted data.
- The receiving device re-calculates the error check based on the received data and compares its newly calculated error check to the error check received with the data.
- If the two error checks match, everything is fine. If they do not match, an error has occurred.

Several error detection techniques of varying degrees of complexity have been developed. The following error detection techniques will be discussed further:

- Parity (VRC)
- Longitudinal Redundancy Checks (LRC)
- Checksums
- Cyclic Redundancy Checks (CRC)

Parity Parity, also known as a (**vertical redundancy check or VRC**), is the simplest error-detection technique. Parity works by adding an error check bit to each character. For example, since the ASCII 7-bit code for the letter A is 1000001, a parity bit is added as the eighth bit. Whether this bit should be a 0 or a 1 depends on whether odd or even parity has been defined or agreed upon in advance by communicating devices. These devices can also agree to data transmission with no parity. Examples of the letter A with odd and even parity are illustrated in Figure 2-29.

As can be seen in Figure 2-29, when the letter A is transmitted with odd parity, the parity bit is set to 1 so that there is an odd number of ones in the 8-bit character. Conversely, when even parity is used, the parity bit is set to 0 so that there is an even

Letter "A" with odd and even parity

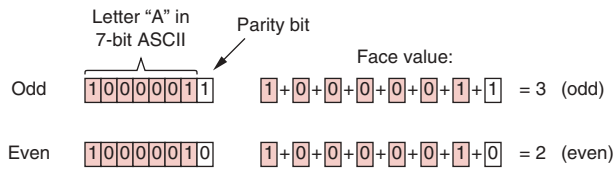


Figure 2-29 Odd and Even Parity

number of ones in the 8-bit character. If even parity is chosen and the receiving computer detects an off number of ones in the character, then it knows that a transmission error has occurred.

Parity checking has a limitation. It can only detect odd numbers of bit-substitution errors. If an even number of bit-substitution errors occur, the received character will pass the parity check, even though the character received was not the character sent. Figure 2-30 illustrates parity checking's inability to detect even numbers of bit-substitution errors within the same character.

As can be seen in Figure 2-30, the received character has an even number of ones so that the receiving computer thinks everything is fine. However, a letter Q was received instead of the letter A that was sent.

Longitudinal Redundancy Checks Longitudinal redundancy checks (LRC) seek to overcome the weakness of simple, bit-oriented, one-directional parity checking. One can think of LRC as adding a second dimension to parity. Figure 2-31 illustrates LRC with even parity in two dimensions.

As can be seen in Figure 2-31, LRC is a block-oriented parity-checking mechanism that adds an entire parity character following a block of data characters. The bits of the parity character are set to establish property parity for each bit position within the block of data characters. In the example illustrated in Figure 2-31 both the parity bits of individual characters (VRC) as well the parity bits of the LRC character would be checked to be sure that even parity existed in both directions.

Checksums Checksums are also block-oriented error detection characters added to a block of data characters. However, rather than checking the number of ones in the block, as was the case with the LRC, a checksum is calculated by adding the decimal

Multiple bit errors/even parity

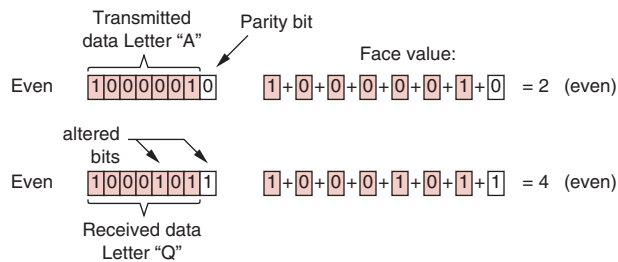


Figure 2-30 Parity Checking's Inability to Detect Multiple Bit Errors

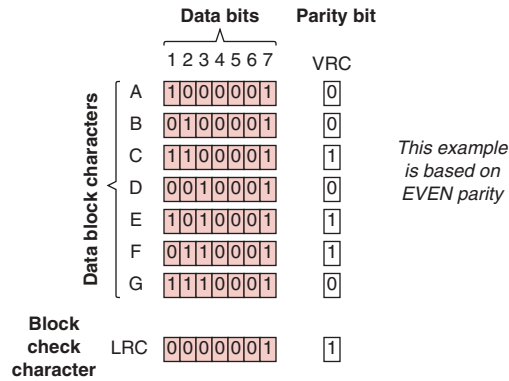


Figure 2-31 Longitudinal Redundancy Checks

face values of all of the characters sent in a given data block and sending only the least significant byte of that sum. The receiving computer generates its own checksum based on the data it has received, and compares the locally calculated checksum with the transmitted checksum.



CHECKSUM CALCULATION

The formula for calculating a checksum is as follows:

- Add the ASCII decimal face value of each of the 128 characters in the block.
- Divide this number by 255.
- Determine the remainder. The remainder is the checksum character to be transmitted to and verified by the receiving computer.

In order to understand the inner workings of this formula, it is necessary to further understand the ASCII table, first introduced in Figure 2-2. Every value in the ASCII table has a decimal equivalent computed by transforming the ones and zeroes of any character, into their Base2 column values and adding the decimal values of those Base2 columns in which ones appear.

Example:

The 7-bit ASCII code for the Capital letter A is 1000001.
Now assign Base 2 place values to each column.

Power of 2	7	6	5	4	3	2	1	0
Decimal Value	128	64	32	16	8	4	2	1
Letter "A" Code		1	0	0	0	0	0	1

There is a 1 in the 64 column and a 1 in the 1 column. Adding these place values, yields 64 + 1, or 65 (Base 10), or decimal 65. Therefore, the ASCII decimal face value of the capital letter A is 65. Some ASCII and EBCDIC tables supply only decimal values rather than binary values, as illustrated in Figures 2-2 and 2-3.

If 128 capital As were transferred as a single block of data, the total ASCII face value of the block would be $128 \times 65 = 8,320$. Dividing the total ASCII face value by 255 yields $8,320/255 = 32 \text{ r } 160$. So the remainder is 160. This remainder is also known as the *least significant byte*.

In order to represent the remainder (160) as a single checksum character in binary format, use the decimal values chart of the powers of 2.

Power of 2	7	6	5	4	3	2	1	0
Decimal Value	128	64	32	16	8	4	2	1
Binary Value of 160	1	0	1	0	0	0	0	0

Thus, the transmitted checksum character would be 10100000.

When dividing our total ASCII face value by 255, the highest remainder possible would be 254. Decimal 254 would be represented in binary and transmitted in a single checksum character (8 bits) as 11111110.

Cyclic Redundancy Checks Cyclic redundancy checks (CRC) seek to improve on the error detection capability of checksums and LRCs. A CRC is really a more sophisticated form of a checksum. In order to understand CRCs, binary division is required. In cyclic redundancy checking, the entire message block of ones and zeroes, even if its 1,000 bits long, is treated as a single, gigantic binary number. This gigantic string of ones and zeroes is divided by a predetermined prime binary number one bit longer than the desired number of bits in the CRC value. Remembering that if these divisors are prime (only divisible by 1 and themselves) they will produce a remainder one bit shorter than themselves. Common lengths for CRCs are 16 and 32 bits. A 16-bit CRC uses a 17-bit divisor and a 32-bit CRC uses a 33-bit divisor.

This remainder is then attached to the actual data message (the original string of ones and zeroes to be transmitted) and transmitted to the receiving device where the received data string is again divided by the same divisor. The remainder calculated at the receiving device is compared to the remainder received from the transmitting device.

Using this technique, a error bursts up to one bit less than the CRC (15 bits for a 16 bit CRC and 31 bits for a 32 bit CRC) can be detected 100 percent of the time, and larger error bursts at $100 - 1/2^{\text{(number of bits in the CRC)}}$ percent of the time. Using this formula to compute the overall percentage for a 16-bit CRC yields an error detection rate of 99.99984742 percent.

Error Correction

Once it is understood how data transmission errors can be detected, they must be reliably and efficiently corrected. In simple terms, error correction amounts to this:

- The receiving device detecting the error and requesting a re-transmission
- The transmitting device retransmitting the data containing the error

The differences in sophistication between the various error correcting protocols are centered on a few variables:

- How is the retransmission requested?
- How much data must be retransmitted?
- How is retransmission time minimized?

Automatic Retransmission Request (ARQ) ARQ or **Automatic Retransmission Request** is a general term that really describes the overall process of error detection using one of the previously described methods, as well as the subsequent automatic request for retransmission of that data. As noted above, the actual request for retransmission can occur in different ways.

Discrete ARQ—ACK/NAK Discrete ARQ is sometimes also known as “*stop and wait*” ARQ. In a protocol using Discrete ARQ, the transmitting device sends a block of data and waits until the remote receiving device does the following:

- Receives the data
- Tests for errors
- Sends an **acknowledgment (ACK)** character back to the transmitting device if the transmitted data were free of errors or
- Sends a **negative acknowledgment (NAK)** character if the data were not successfully received

After waiting for this ACK/NAK, the transmitting device does the following:

- Sends the next block of data if an ACK was received or
- Retransmits the original block of data if a NAK was received

This entire process repeats itself until the conclusion of the data transmission session. Using this technique the transmitting device spends a significant amount of time idly waiting for either an ACK or a NAK to be sent by the receiving device. This is inefficient, so other ARQ methods have been devised that take a different, more efficient approach to error detection and correction.

Continuous ARQ A variation of ARQ known as **continuous ARQ**, eliminates the requirement for the transmitting device to wait for an ACK or a NAK after transmitting each block before transmitting the next block of data, eliminating a great deal of idle time and increasing overall data throughput.

With continuous ARQ, also known as a **sliding window protocol**, a **sequence number** is appended to each block of data transmitted. The transmitting device continuously transmits blocks of data without waiting for ACK or NAK from the receiving device after each block of data sent. However, there is often a block limit to prevent a device from transmitting indefinitely without having received either an ACK or a NAK. The receiving device still checks each block of data for errors, just as it did with discrete ARQ. However, when the receiving device detects an error in a block of data, it sends the sequence number along with the NAK back to the transmitting device. The transmitting device now knows which particular block of data was received in error, slides the transmission window back to the NAK'd block, and resumes transmission from that point.

Selective ARQ Continuous ARQ's requirement to retransmit *all* blocks of data from the NAK'd block forward is more efficient than discrete ARQ but can still be made more efficient. **Selective ARQ** increases efficiency by requiring only the retransmission of the blocks received in error rather than the block in error and all subsequent blocks.

Flow Control

When a data communication device receives data it is placed in a special place in memory known as buffer memory (or simply a **buffer**) until it can be sent to the attached destination device. To ensure the data are sent to the destination device in the proper order, they are stored in the order they were sent. The amount of memory available in the buffer is finite: once the buffer is full no more data can be stored and any data that come in will be discarded. The process of discarding data because the buffer is full is known as a buffer **overflow**, although data communication professionals will sometimes refer to it as placing data in the **bit bucket**. Obviously, discarding data because the buffer is full is an undesirable situation.

To prevent buffer overflows, the receiving device sends a signal to the sending device that says, in effect, “shut up for a minute while I catch up.” This is not unlike students raising their hand to ask the professor to wait before changing slides or erasing the board so they can finish taking notes. When it occurs in a data communication application, this process is known as **flow control**. The flow control software constantly monitors the amount of free space available in buffer memory and tells the sending device to stop sending data when there is insufficient storage space. When the buffer once again has room, the sending device is told to resume transmitting.

SUMMARY

Key concepts used in all types of data communication technologies were introduced in this chapter.

Data must be in a digital form before it can be sent across a data network. CODECs and character encoding standards such as ASCII and UNICODE are used to convert analog data into a digital format prior to transmission.

The two main types of data transmission are parallel and serial. In parallel transmission, multiple bits are sent concurrently across multiple signal wires, while in serial transmission the bits are sent one at a time across one signaling wire. Parallel transmission is typically used inside a computer while serial transmission is typically used between computers.

Although all data sent across a data communications network must be in a digital form, it is often sent using analog signal signaling. For the digital signal to be sent across an analog transmission link it must be modulated using techniques such as AM, FM, PM, and QAM.

A data stream must also be packetized into chunks for transmission. These chunks are then embedded into protocols in a process known as encapsulation before being sent across the communications link. At the destination the protocols are un-encapsulated and the data de-packetized before being delivered to the destination application.

To enable efficient use of the communications link, multiplexing is often used to allow multiple communication sessions to share a single link. The most commonly used multiplexing techniques are FDM, TDM, and STDM.

When building a network, some means of creating links between the source and destination devices is required. There are two basic approaches to this: circuit switching and packet switching. In circuit switching, a physical circuit is connected between the devices. In packet switching, the packetized data between the devices is routed across shared circuits to the destination. In modern networks, packet switching is used almost exclusively.

To ensure accurate reception of transmitted data, error control is implemented. There are three basic parts of error control: error prevention, error detection, and error correction. Error prevention includes activities such as line conditioning to lessen the chance of an error occurring, error detection includes technologies such as parity and CRCs to allow errors to be

detected, and error correction includes the processes used to request re-transmissions of packets that contain errors.

Flow control is used to ensure that the data is not sent faster than the receiving device can process it. Flow control techniques provide a means for the receiver to tell the sender to wait before transmitting an additional data.

KEY TERMS

ACK	CRC	parity
adaptive protocols	ARQ	phase
amplitude	data sinks	phase modulation
amplitude modulation	datagram	polling
ASCII	discrete ARQ	principle of shifting bottlenecks
asynchronous transmission	duplex	PSK
baud	EBCDIC	public data networks
baud rate	flow control	QAM
bit	frequency	QPSK
bit bucket	frequency division multiplexing	sequence number
bps	frequency modulation	serial transmission
buffer	FSK	signal to noise ratio
buffer overflow	guardbands	sliding window
byte	intersymbol interference	statistical time division multiplexing
carrier wave	longitudinal redundancy check	synchronous transmission
central clock	modem	time division multiplexing
character encoding	modulation	time slot
checksum	multiplexing	transmission rate
circuit switching	NAK	turnaround time
composite message frame	octet	UNICODE
connectionless communication	packet switches	vertical redundancy check
connection-oriented communication	packet switching	wavelength
constellation points	packetization	wavelength division multiplexing
continuous ARQ	PAD	
	parallel transmission	

REVIEW QUESTIONS

1. What are the three major character encoding standards?
2. Why isn't there a single encoding standard?
3. What is character encoding and why is it necessary?
4. What is the importance of encoding standards such as Unicode/ISO 10646?
5. What is a disadvantage of encoding standards such as Unicode/ISO 10646?
6. What is a bit and how is it represented within a computer?
7. What are the primary differences between serial and parallel transmission?
8. Which type of transmission (serial or parallel) is most often used in data communications and why?
9. What is a UART and what role does it play in the overall process of getting data from a local PC to a remote PC?
10. What is the difference between analog transmission and digital transmission?
11. What is the difference between asynchronous transmission and synchronous transmission? Which is more efficient?
12. Modem is actually a contraction for which two words?
13. What is a carrier wave?
14. What three characteristics of a carrier wave can be varied?
15. How are the number of detectable events related to the baud rate and bits per second (bps)?
16. What is the importance of signal to noise ratio in terms of modem design and operation?
17. What is the advantage of varying more than one characteristic of a carrier wave?
18. What is the difference between full-duplex and half-duplex data transmission?
19. What is the difference between bps and baud rate?
20. Explain in simple terms how a circuit-switched or dial-up call is established.
21. Complete an I-P-O chart illustrating the required functionality of a modem.
22. What role does a carrier wave play in data transmission over an analog communications link?
23. What is the relationship between wavelength and frequency?
24. What is a signalling event?
25. Which transmission methodology requires an external clocking source? Why?
26. How are handshaking and turnaround time related to full-duplex and half-duplex transmission?
27. Give an example of how physical interfaces or connectors don't necessarily imply a transmission protocol.
28. What is a constellation point?
29. What is intersymbol interference and what impact does it have on modulation scheme design?
30. Why are standards important when it comes to data compression?
31. What is packetization?
32. Why is packetization important in a data communication network?
33. What is multiplexing?
34. Explain frequency division multiplexing.
35. Explain time division multiplexing.
36. Explain statistical time division multiplexing.
37. Which multiplexing technique is most efficient?
38. Why is switching a requirement for a data communication network?
39. Compare and contrast circuit and packet switching.
40. What is a packet assembler/dis-assembler and what is its purpose?
41. What is a datagram?
42. Why are connectionless networks considered unreliable?
43. Elaborate on the relationship between Error Prevention, Error Detection, and Error Correction.
44. What is the difference between discrete ARQ and continuous ARQ?
45. What inefficiency is inherent in a data transmission session utilizing discrete ARQ over a full duplex circuit?
46. Explain how multiple bit errors can remain undetected using simple parity checking.
47. Explain how LRC overcomes parity checking's inability to detect multiple bit errors.
48. Explain in simple terms how checksums and CRCs detect transmission errors.
49. What is meant by a sliding window protocol?
50. How are block sequence numbers related to sliding window protocols?
51. What role does buffer memory play in the implementation of sliding window protocols?
52. Why is flow control important?