CHAPTER 6

Error Detection and Correction

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Explain the basic mechanism for the use of error-detecting codes.
- Present an overview of the Internet checksum.
- Understand the operation of the cyclic redundancy check.
- Define Hamming distance.
- Explain the basic principles for forward error correction using a block code.

In earlier chapters, we talked about transmission impairments and the effect of data rate and signal-to-noise ratio on bit error rate. Regardless of the design of the transmission system, there will be errors, resulting in the change of one or more bits in a transmitted block of data.

Three approaches are in common use for coping with data transmission errors:

- Error-detection codes
- Error-correction codes, also called forward error correction (FEC) codes
- Automatic repeat request (ARQ) protocols

All three approaches are applied to individual blocks of data, called frames, packets, or cells, depending on the protocol used for data exchange. The first two approaches, which are the subject of this chapter, use **redundancy**. In essence, for error detection and error correction, additional bits, which are a function of the data bits, are appended to the data bits by the sender. These redundant bits are used by the receiver for the purpose of error detection or correction.

An error-detection code simply detects the presence of an error. Typically, such codes are used in conjunction with a protocol at the data link or transport level (see Figure 2.3) that uses an ARQ scheme. With an ARQ scheme, a receiver discards a block of data in which an error is detected and the transmitter retransmits that block of data. FEC codes are designed not just to detect but correct errors, avoiding the need for retransmission. FEC schemes are frequently used in wireless transmission, where retransmission schemes are highly inefficient and error rates may be high.

This chapter examines error-detecting and error-correcting codes. ARQ protocols are discussed in Chapter 7. After a brief discussion of the

distinction between single-bit errors and burst errors, this chapter describes three approaches to error detection: the use of parity bits, the Internet checksum technique, and the cyclic redundancy check (CRC). The CRC error-detecting code is used in a wide variety of applications. Because of its complexity, we present three different approaches to describing the CRC algorithm.

For error correction, this chapter provides an overview of the general principles of error-correcting codes. Specific examples are provided in Chapter 16.

6.1 TYPES OF ERRORS

In digital transmission systems, an error occurs when a bit is altered between transmission and reception; that is, a binary 1 is transmitted and a binary 0 is received, or a binary 0 is transmitted and a binary 1 is received. Two general types of errors can occur: single-bit errors and burst errors. A single-bit error is an isolated error condition that alters one bit but does not affect nearby bits. A burst error of length B is a contiguous sequence of B bits in which the first and last bits and any number of intermediate bits are received in error. More precisely, IEEE Std 100 and ITU-T Recommendation Q.9 both define an error burst as follows:

> **Error burst:** A group of bits in which two successive erroneous bits are always separated by less than a given number x of correct bits. The last erroneous bit in the burst and the first erroneous bit in the following burst are accordingly separated by x correct bits or more.

Thus, in an error burst, there is a cluster of bits in which a number of errors occur, although not necessarily all of the bits in the cluster suffer an error. Figure 6.1 provides an example of both types of errors.

A single-bit error can occur in the presence of white noise, when a slight random deterioration of the signal-to-noise ratio is sufficient to confuse the receiver's decision of a single bit. Burst errors are more common and more difficult to deal with. Burst errors can be caused by impulse noise, which was described in Chapter 3. Another cause is fading in a mobile wireless environment; fading is described in Chapter 10.

Note that the effects of burst errors are greater at higher data rates.

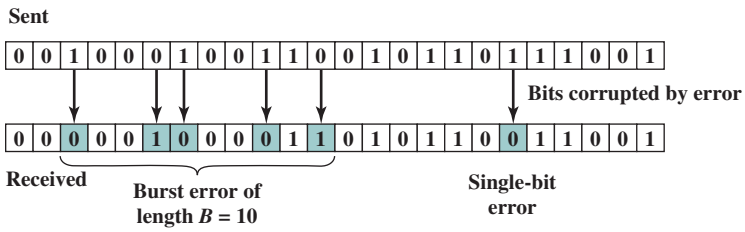


Figure 6.1 Burst and Single-Bit Errors

EXAMPLE 6.1 An impulse noise event or a fading event of 1 μs occurs. At a data rate of 10 Mbps, there is a resulting error burst of 10 bits. At a data rate of 100 Mbps, there is an error burst of 100 bits.

6.2 ERROR DETECTION

Regardless of the design of the transmission system, there will be errors, resulting in the change of one or more bits in a transmitted frame. In what follows, we assume that data are transmitted as one or more contiguous sequences of bits, called frames. We define these probabilities with respect to errors in transmitted frames:

P_b: Probability that a bit is received in error; also known as the bit error rate (BER)

P_1: Probability that a frame arrives with no bit errors

P_2: Probability that, with an error-detecting algorithm in use, a frame arrives with one or more undetected errors

P_3: Probability that, with an error-detecting algorithm in use, a frame arrives with one or more detected bit errors but no undetected bit errors

First consider the case in which no means are taken to detect errors. Then the probability of detected errors (P_3) is zero. To express the remaining probabilities, assume the probability that any bit is in error (P_b) is constant and independent for each bit. Then we have

$$P_1 = (1 - P_b)^F$$
$$P_2 = 1 - P_1$$

where F is the number of bits per frame. In words, the probability that a frame arrives with no bit errors decreases when the probability of a single bit error increases, as you would expect. Also, the probability that a frame arrives with no bit errors decreases with increasing frame length; the longer the frame, the more bits it has and the higher the probability that one of these is in error.

EXAMPLE 6.2 A defined objective for ISDN (integrated services digital network) connections is that the BER on a 64-kbps channel should be less than 10^{-6} on at least 90% of observed 1-minute intervals. Suppose now that we have the rather modest user requirement that on average one frame with an undetected bit error should occur per day on a continuously used 64-kbps channel, and let us assume a frame length of 1000 bits. The number of frames that can be transmitted in a day comes out to 5.529×10^6, which yields a desired frame error rate of $P_2 = 1/(5.529 \times 10^6) = 0.18 \times 10^{-6}$. But if we assume a value of P_b of 10^{-6}, then $P_1 = (0.999999)^{1000} = 0.999$ and therefore $P_2 = 10^{-3}$, which is about three orders of magnitude too large to meet our requirement.

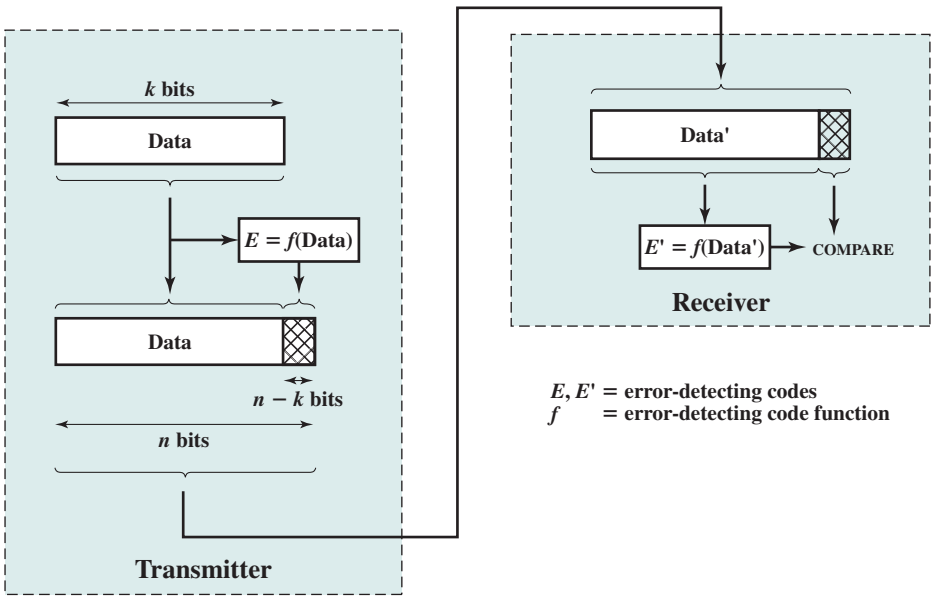


Figure 6.2 Error-Detection Process

This is the kind of result that motivates the use of error-detecting techniques to achieve a desired frame error rate on a connection with a given BER. All of these techniques operate on the following principle (Figure 6.2). For a given frame of bits, additional bits that constitute an **error-detecting code** are added by the transmitter. This code is calculated as a function of the other transmitted bits. Typically, for a data block of k bits, the error-detecting algorithm yields an error-detecting code of $n - k$ bits, where $(n - k) < k$. The error-detecting code, also referred to as the **check bits**, is appended to the data block to produce a frame of n bits, which is then transmitted. The receiver separates the incoming frame into the k bits of data and $(n - k)$ bits of the error-detecting code. The receiver performs the same error-detecting calculation on the data bits and compares this value with the value of the incoming error-detecting code. A detected error occurs if and only if there is a mismatch. Thus P_3 is the probability that a frame contains errors and that the error-detecting scheme will detect that fact. P_2 is known as the residual error rate and is the probability that an error will be undetected despite the use of an error-detecting scheme.

6.3 PARITY CHECK

Parity Bit

The simplest error-detecting scheme is to append a parity bit to the end of a block of data. A typical example is character transmission, in which a parity bit is attached to each 7-bit IRA character. The value of this bit is selected so that the character has an even number of 1s (even parity) or an odd number of 1s (odd parity).

EXAMPLE 6.3 If the transmitter is transmitting an IRA character G (1110001) and using odd parity, it will append a 1 and transmit 11110001.[1] The receiver examines the received character and, if the total number of 1s is odd, assumes that no error has occurred. If one bit (or any odd number of bits) is erroneously inverted during transmission (e.g., 11100001), then the receiver will detect an error.

Note, however, that if two (or any even number) of bits are inverted due to error, an undetected error occurs. Typically, even parity is used for synchronous transmission and odd parity for asynchronous transmission.

The use of the parity bit is not foolproof, as noise impulses are often long enough to destroy more than one bit, particularly at high data rates.

Two-Dimensional Parity Check

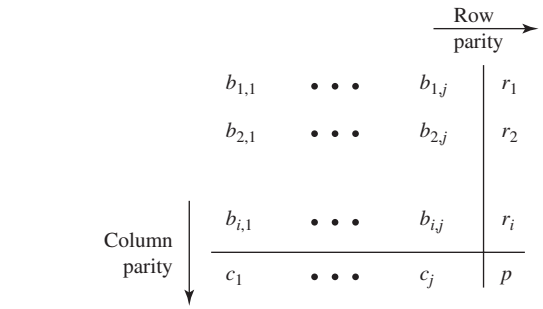
The two-dimensional parity scheme, illustrated in Figure 6.3, is more robust than the single parity bit. The string of data bits to be checked is arranged in a two-dimensional array. Appended to each row i is an even parity bit r_i for that row, and appended to each column j is an even parity bit c_j for that column. An overall parity bit p completes the matrix. Thus the error-detecting code consists of $i + j + 1$ parity bits.

In this scheme, every bit participates in two parity checks. As with a simple parity bit, any odd number of bit errors is detected.

EXAMPLE 6.4 Figure 6.3b shows a string of 20 data bits arranged in a 4×5 array, with the parity bits calculated, to form a 5×6 array. When a single-bit error occurs in Figure 6.3c, both the corresponding row and column parity bits now indicate an error. Furthermore, the error can be determined to be at the intersection of that row and column. Thus it is possible to not only detect but also correct the bit error.

If an even number of errors occur in a row, the errors are detected by the column parity bits. Similarly, if an even number of errors occur in a column, the errors are detected by the row parity bits. However, any pattern of four errors forming a rectangle, as shown in Figure 6.3d, is undetectable. If the four circled bits change value, the corresponding row and column parity bits do not change value.

[1]Recall from our discussion in Section 5.1 that the least significant bit of a character is transmitted first and that the parity bit is the most significant bit.



(a) Parity calculation

0	1	1	1	0	1
0	1	1	1	0	1
0	1	0	0	0	1
0	1	0	1	1	1
0	0	0	1	1	0

(b) No errors

0	1	1	1	0	1
0	(0)	1	1	0	1
0	1	0	0	0	1
0	1	0	1	1	1
0	0	0	1	1	0

Row parity error

Column parity error

(c) Correctable single-bit error

0	1	1	1	1	1	0	1
0	(0)	1	1	0	(1)	1	0
0	0	1	1	0	0	1	1
0	(0)	0	0	0	(0)	0	0
1	0	1	1	1	1	1	0
1	1	0	0	0	1	1	0

(d) Uncorrectable error pattern

Figure 6.3 A Two-Dimensional Even Parity Scheme

6.4 THE INTERNET CHECKSUM

The Internet checksum is an error-detecting code used in many Internet standard protocols, including IP, TCP, and UDP. The calculation makes use of the ones-complement operation and ones-complement addition.[2] To perform the **ones-complement operation** on a set of binary digits, replace 0 digits with 1 digits and 1 digits with 0 digits. The **ones-complement addition** of two binary integers of equal bit length is performed as follows:

1. The two numbers are treated as unsigned binary integers and added.
2. If there is a carry out of the leftmost bit, add 1 to the sum. This is called an *end-around carry*.

[2]See Appendix H for a further discussion of ones-complement arithmetic.

Here are two examples:

$$\begin{array}{r} 0011 \\ +1100 \\ \hline 1111 \end{array} \qquad \begin{array}{r} 1101 \\ +1011 \\ \hline 11000 \\ +\quad 1 \\ \hline 1001 \end{array}$$

Typically, the checksum is included as a field in the header of a protocol data unit, such as in IP datagram. To compute the checksum, the checksum field is first set to all zeros. The checksum is then calculated by performing the ones-complement addition of all the words in the header, and then taking the ones-complement operation of the result. This result is placed in the checksum field.

To verify a checksum, the ones-complement sum is computed over the same set of octets, including the checksum field. If the result is all 1 bits (−0 in ones-complement arithmetic), the check succeeds.

EXAMPLE 6.5 Consider a header that consists of 10 octets, with the checksum in the last two octets (this does not correspond to any actual header format) with the following content (in hexadecimal):

00 01 F2 03 F4 F5 F6 F7 00 00

Note that the checksum field is set to zero.

Figure 6.4a shows the results of the calculation. Thus, the transmitted packet is 00 01 F2 03 F4 F5 F6 F7 0D. Figure 6.4b shows the calculation carried out by the receiver on the entire data block, including the checksum. The result is a value of all ones, which verifies that no errors have been detected.

Partial sum	0001 F203 F204
Partial sum	F204 F4F5 1E6F9
Carry	E6F9 1 E6FA
Partial sum	E6FA F6F7 1DDF1
Carry	DDF1 1 DDF2
Ones complement of the result	220D

(a) Checksum calculation by sender

Partial sum	0001 F203 F204
Partial sum	F204 F4F5 1E6F9
Carry	E6F9 1 E6FA
Partial sum	E6FA F6F7 1DDF1
Carry	DDF1 1 DDF2
Partial sum	DDF2 220D FFFF

(b) Checksum verification by receiver

Figure 6.4 Example of Internet Checksum

The Internet checksum provides greater error-detection capability than a parity bit or two-dimensional parity scheme but is considerably less effective than the cyclic redundancy check (CRC), discussed next. The primary reason for its adoption in Internet protocols is efficiency. Most of these protocols are implemented in software and the Internet checksum, involving simple addition and comparison operations, causes very little overhead. It is assumed that at the lower link level, a strong error-detection code such as CRC is used, and so the Internet checksum is simply an additional end-to-end check for errors.

6.5 CYCLIC REDUNDANCY CHECK (CRC)

One of the most common, and one of the most powerful, error-detecting codes is the cyclic redundancy check, which can be described as follows. Given a k-bit block of bits, or message, the transmitter generates an $(n - k)$-bit sequence, known as a **frame check sequence (FCS)**, such that the resulting frame, consisting of n bits, is exactly divisible by some predetermined number. The receiver then divides the incoming frame by that number and, if there is no remainder, assumes there was no error.[3]

To clarify this, we present the procedure in three equivalent ways: modulo 2 arithmetic, polynomials, and digital logic.

Modulo 2 Arithmetic

Modulo 2 arithmetic uses binary addition with no carries, which is just the exclusive-OR (XOR) operation. Binary subtraction with no carries is also interpreted as the XOR operation: For example

$$\begin{array}{r} 1111 \\ +1010 \\ \hline 0101 \end{array} \qquad \begin{array}{r} 1111 \\ -0101 \\ \hline 1010 \end{array} \qquad \begin{array}{r} 11001 \\ \times\ 11 \\ \hline 11001 \\ 11001 \\ \hline 101011 \end{array}$$

Now define

$T = n$-bit frame to be transmitted

$D = k$-bit block of data, or message, the first k bits of T

$F = (n - k)$-bit FCS, the last $(n - k)$ bits of T

$P =$ pattern of $n - k + 1$ bits; this is the predetermined divisor

We would like T/P to have no remainder. It should be clear that

$$T = 2^{n-k}D + F$$

[3]This procedure is slightly different from that of Figure 6.2. As shall be seen, the CRC process could be implemented as follows. The receiver could perform a division operation on the incoming k data bits and compare the result to the incoming $(n - k)$ check bits.

That is, by multiplying D by 2^{n-k}, we have in effect shifted it to the left by $n-k$ bits and padded out the result with zeroes. Adding F yields the concatenation of D and F, which is T. We want T to be exactly divisible by P. Suppose that we divide $2^{n-k}D$ by P:

$$\frac{2^{n-k}D}{P} = Q + \frac{R}{P} \tag{6.1}$$

There is a quotient and a remainder. Because division is modulo 2, the remainder is always at least one bit shorter than the divisor. We will use this remainder as our FCS. Then

$$T = 2^{n-k}D + R \tag{6.2}$$

Does this R satisfy our condition that T/P has no remainder? To see that it does, consider:

$$\frac{T}{P} = \frac{2^{n-k}D + R}{P} = \frac{2^{n-k}D}{P} + \frac{R}{P}$$

Substituting Equation (6.1), we have

$$\frac{T}{P} = Q + \frac{R}{P} + \frac{R}{P}$$

However, any binary number added to itself modulo 2 yields zero. Thus

$$\frac{T}{P} = Q + \frac{R + R}{P} = Q$$

There is no remainder, and therefore T is exactly divisible by P. Thus, the FCS is easily generated: Simply divide $2^{n-k}D$ by P and use the $(n-k)$-bit remainder as the FCS. On reception, the receiver will divide T by P and will get no remainder if there have been no errors.

EXAMPLE 6.6

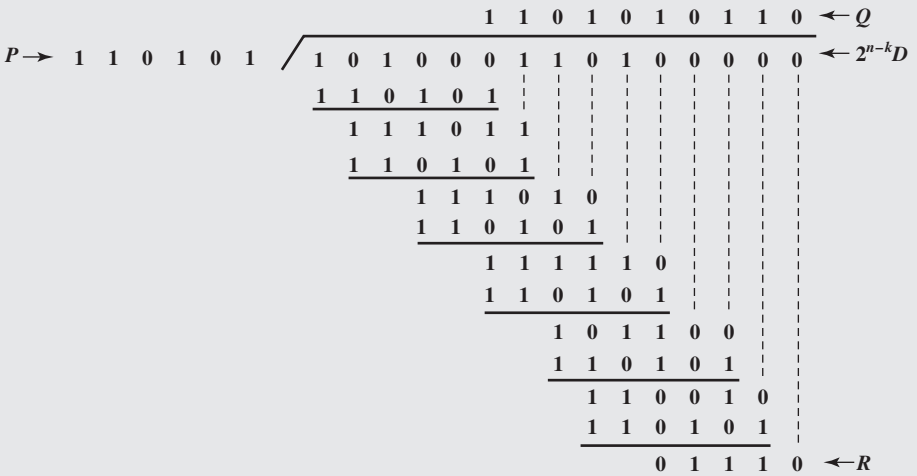
1. Given

$$\begin{aligned} \text{Message } D &= 1010001101 \text{ (10 bits)} \\ \text{Pattern } P &= 110101 \text{ (6 bits)} \\ \text{FCS } R &= \text{ to be calculated (5 bits)} \end{aligned}$$

Thus, $n = 15$, $k = 10$, and $(n-k) = 5$.

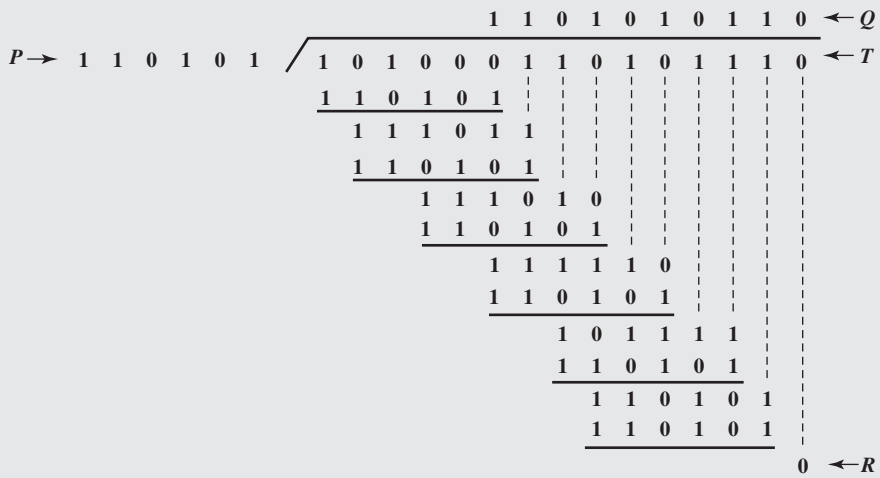
2. The message is multiplied by 2^5, yielding 101000110100000.

3. This product is divided by P:



4. The remainder is added to 2^5D to give $T = 101000110101110$, which is transmitted.

5. If there are no errors, the receiver receives T intact. The received frame is divided by P:



Because there is no remainder, it is assumed that there have been no errors.

The pattern P is chosen to be one bit longer than the desired FCS, and the exact bit pattern chosen depends on the type of errors expected. At minimum, both the high- and low-order bits of P must be 1.

There is a concise method for specifying the occurrence of one or more errors. An error results in the reversal of a bit. This is equivalent to taking the XOR of the

bit and 1 (modulo 2 addition of 1 to the bit): $0 + 1 = 1$; $1 + 1 = 0$. Thus, the errors in an n-bit frame can be represented by an n-bit field with 1s in each error position. The resulting frame T_r can be expressed as

$$T_r = T \oplus E$$

where

T = transmitted frame
E = error pattern with 1s in positions where errors occur
T_r = received frame
$\oplus$ = bitwise exclusive-OR (XOR)

If there is an error ($E \neq 0$), the receiver will fail to detect the error if and only if T_r is divisible by P, which is equivalent to E divisible by P. Intuitively, this seems an unlikely occurrence.

Polynomials

A second way of viewing the CRC process is to express all values as polynomials in a dummy variable X, with binary coefficients. The coefficients correspond to the bits in the binary number. Thus, for $D = 110011$, we have $D(X) = X^5 + X^4 + X + 1$, and for $P = 11001$, we have $P(X) = X^4 + X^3 + 1$. Arithmetic operations are again modulo 2. The CRC process can now be described as

$$\frac{X^{n-k}D(X)}{P(X)} = Q(X) + \frac{R(X)}{P(X)}$$

$$T(X) = X^{n-k}D(X) + R(X)$$

Compare these equations with Equations (6.1) and (6.2).

EXAMPLE 6.7 Using the preceding example, for $D = 1010001101$, we have $D(X) = X^9 + X^7 + X^3 + X^2 + 1$, and for $P = 110101$, we have $P(X) = X^5 + X^4 + X^2 + 1$. We should end up with $R = 01110$, which corresponds to $R(X) = X^3 + X^2 + X$. Figure 6.5 shows the polynomial division that corresponds to the binary division in the preceding example.

An m-bit CRC is typically generated by a polynomial of the form

$$P(X) = q(X) \text{ or}$$
$$P(X) = (X + 1)q(X)$$

Where $q(X)$ is a special type of polynomial called a primitive polynomial.[4]

[4]A document at box.com/dcc10e provides an explanation of primitive polynomials.

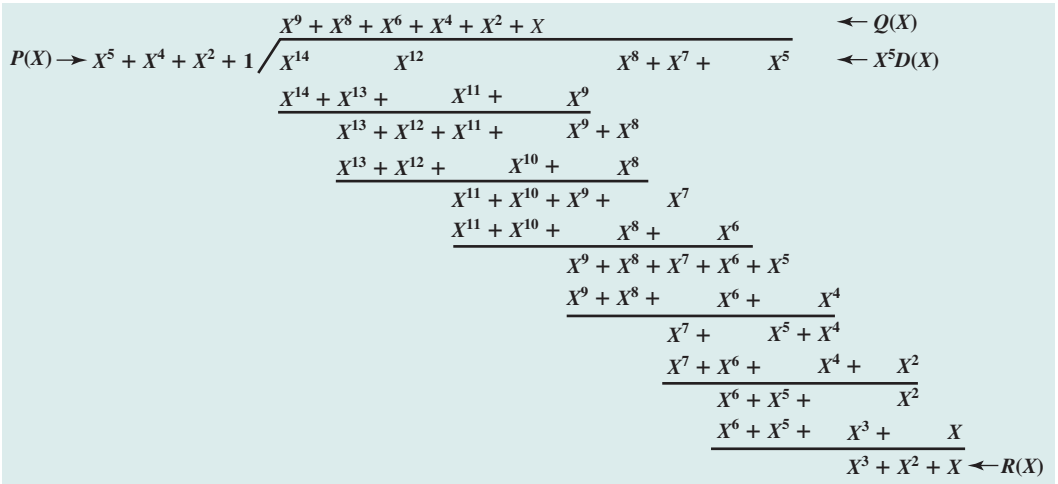


Figure 6.5 Example of Polynomial Division

An error $E(X)$ will only be undetectable if it is divisible by $P(X)$. It can be shown [PETE61, RAMA88] that all of the following errors are not divisible by a suitably chosen $P(X)$ and hence are detectable:

- All single-bit errors, if $P(X)$ has more than one nonzero term
- All double-bit errors, as long as $P(X)$ is of one of the two forms shown above
- Any odd number of errors, as long as $P(X)$ contains a factor $(X + 1)$
- Any burst error for which the length of the burst is less than or equal to $n - k$; that is, less than or equal to the length of the FCS
- A fraction of error bursts of length $n - k + 1$; the fraction equals $1 - 2^{-(n-k-1)}$
- A fraction of error bursts of length greater than $n - k + 1$; the fraction equals $1 - 2^{-(n-k)}$

In addition, it can be shown that if all error patterns are considered equally likely, then for a burst error of length $r + 1$, the probability of an undetected error ($E(X)$ is divisible by $P(X)$) is $1/2^{r-1}$, and for a longer burst, the probability is $1/2^r$, where r is the length of the FCS.

Four versions of $P(X)$ are widely used:

$$\begin{aligned}
\text{CRC-12} &= X^{12} + X^{11} + X^3 + X^2 + X + 1 = (X + 1)(X^{11} + X^2 + 1) \\
\text{CRC-ANSI} &= X^{16} + X^{15} + X^2 + 1 = (X + 1)(X^{15} + X + 1) \\
\text{CRC-CCITT} &= X^{16} + X^{12} + X^5 + 1 = (X + 1)(X^{15} + X^{14} + X^{13} + X^{12} \\
&\quad + X^4 + X^3 + X^2 + X + 1) \\
\text{IEEE-802} &= X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 \\
&\quad + X^7 + X^5 + X^4 + X^2 + X + 1
\end{aligned}$$

The CRC-12 system is used for transmission of streams of 6-bit characters and generates a 12-bit FCS. Both CRC-16 and CRC-CCITT are popular for 8-bit characters

in the United States and Europe, respectively, and both result in a 16-bit FCS. This would seem adequate for most applications, although CRC-32 is specified as an option in some point-to-point synchronous transmission standards and is used in IEEE 802 LAN standards.

Digital Logic

The CRC process can be represented by, and indeed implemented as, a dividing circuit consisting of XOR gates and a shift register. The shift register is a string of 1-bit storage devices. Each device has an output line, which indicates the value currently stored, and an input line. At discrete time instants, known as clock times, the value in the storage device is replaced by the value indicated by its input line. The entire register is clocked simultaneously, causing a 1-bit shift along the entire register.

The circuit is implemented as follows:

1. The register contains $n - k$ bits, equal to the length of the FCS.
2. There are up to $n - k$ XOR gates.
3. The presence or absence of a gate corresponds to the presence or absence of a term in the divisor polynomial, $P(X)$, excluding the terms 1 and X^{n-k}.

EXAMPLE 6.8 The architecture of a CRC circuit is best explained by first considering an example, which is illustrated in Figure 6.6. In this example, we use:

$$\text{Data } D = 1010001101; \quad D(X) = X^9 + X^7 + X^3 + X^2 + 1$$
$$\textit{Divisor } P = 110101; \quad P(X) = X^5 + X^4 + X^2 + 1$$

which were used earlier in the discussion.

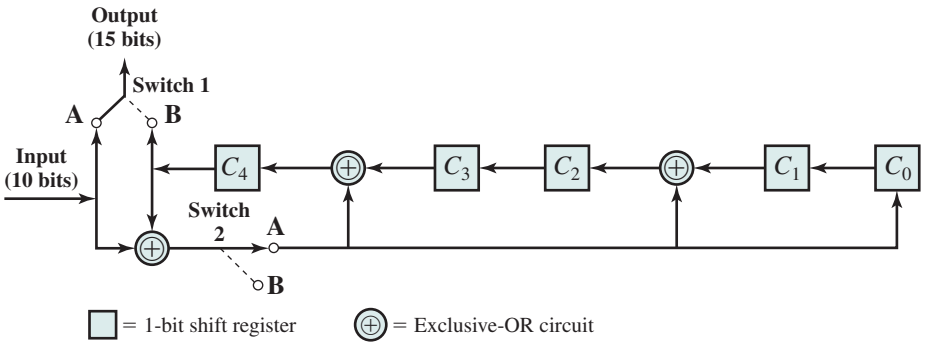
Figure 6.6a shows the shift-register implementation. The process begins with the shift register cleared (all zeros). The message, or dividend, is then entered one bit at a time, starting with the most significant bit. Figure 6.6b is a table that shows the step-by-step operation as the input is applied one bit at a time. Each row of the table shows the values currently stored in the five shift-register elements. In addition, the row shows the values that appear at the outputs of the three XOR circuits. Finally, the row shows the value of the next input bit, which is available for the operation of the next step.

Note that the XOR operation affects C_4, C_2, and C_0 on the next shift. This is identical to the binary long division process illustrated earlier. The process continues through all the bits of the message. To produce the proper output, two switches are used. The input data bits are fed in with both switches in the A position. As a result, for the first 10 steps, the input bits are fed into the shift register and also used as output bits. After the last data bit is processed, the shift register contains the remainder (FCS) (shown shaded). As soon as the last data bit is provided to the shift register, both switches are set to the B position.

This has two effects: (1) all of the XOR gates become simple pass-throughs; no bits are changed, and (2) as the shifting process continues, the 5 CRC bits are output.

At the receiver, the same logic is used. As each bit of M arrives, it is inserted into the shift register. If there have been no errors, the shift register should contain the bit pattern for R at the conclusion of M. The transmitted bits of R now begin to arrive, and the effect is to zero out the register so that, at the conclusion of reception, the register contains all 0s.

Figure 6.7 indicates the general architecture of the shift-register implementation of a CRC for the polynomial $P(X) = \sum_{i=0}^{n-k} A_i X^i$, where $A_0 = A_{n-k} = 1$ and all other A_i equal either 0 or 1.[5]



(a) Shift-register implementation

	C_4	C_3	C_2	C_1	C_0	$C_4 \oplus C_3 \oplus I$	$C_4 \oplus C_1 \oplus I$	$C_4 \oplus I$	I = input	
Initial	0	0	0	0	0	1	1	1	1	
Step 1	1	0	1	0	1	1	1	1	0	
Step 2	1	1	1	1	1	1	1	0	1	
Step 3	1	1	1	1	0	0	0	1	0	
Step 4	0	1	0	0	1	1	0	0	0	Message to be sent
Step 5	1	0	0	1	0	1	0	1	0	
Step 6	1	0	0	0	1	0	0	0	1	
Step 7	0	0	0	1	0	1	0	1	1	
Step 8	1	0	0	0	1	1	1	1	0	
Step 9	1	0	1	1	1	0	1	0	1	
Step 10	0	1	1	1	0					

(b) Example with input of 1010001101

Figure 6.6 Circuit with Shift Registers for Dividing by the Polynomial $X^5 + X^4 + X^2 + 1$

[5]It is common for the CRC register to be shown shifting to the right, which is the reverse of the analogy to binary division. Because binary numbers are usually shown with the most significant bit on the left, a left-shifting register, as is used here, is more appropriate.

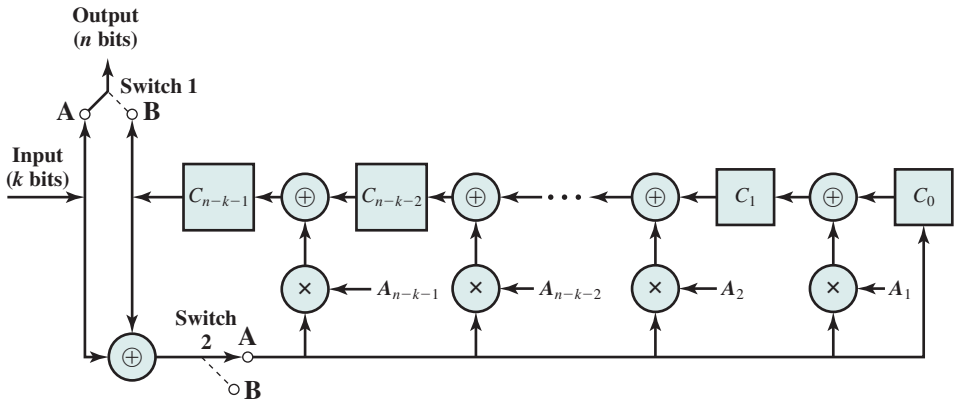


Figure 6.7 General CRC Architecture to Implement Divisor $(1 + A_1X + A_2X + \cdots + A_{n-k-1}X^{n-k-1} + X^{n-k})$

6.6 FORWARD ERROR CORRECTION

Error detection is a useful technique, found in data link control protocols, such as HDLC, and in transport protocols, such as TCP. As explained in Chapter 7, an error-detecting code can be used as part of a protocol that corrects errors in transmitted data by requiring the blocks of data be retransmitted. For wireless applications this approach is inadequate for two reasons:

1. The BER on a wireless link can be quite high, which would result in a large number of retransmissions.
2. In some cases, especially satellite links, the propagation delay is very long compared to the transmission time of a single frame. The result is a very inefficient system. As is discussed in Chapter 7, the common approach to retransmission is to retransmit the frame in error plus all subsequent frames. With a long data link, an error in a single frame necessitates retransmitting many frames.

Instead, it would be desirable to enable the receiver to correct errors in an incoming transmission on the basis of the bits in that transmission. Figure 6.8 shows in general how this is done. On the transmission end, each k-bit block of data is mapped into an n-bit block $(n > k)$ called a **codeword**, using an FEC (forward error correction) encoder. The codeword is then transmitted. During transmission, the signal is subject to impairments, which may produce bit errors in the signal. At the receiver, the incoming signal is demodulated to produce a bit string that is similar to the original codeword but may contain errors. This block is passed through an FEC decoder, with one of four possible outcomes:

- **No errors:** If there are no bit errors, the input to the FEC decoder is identical to the original codeword, and the decoder produces the original data block as output.

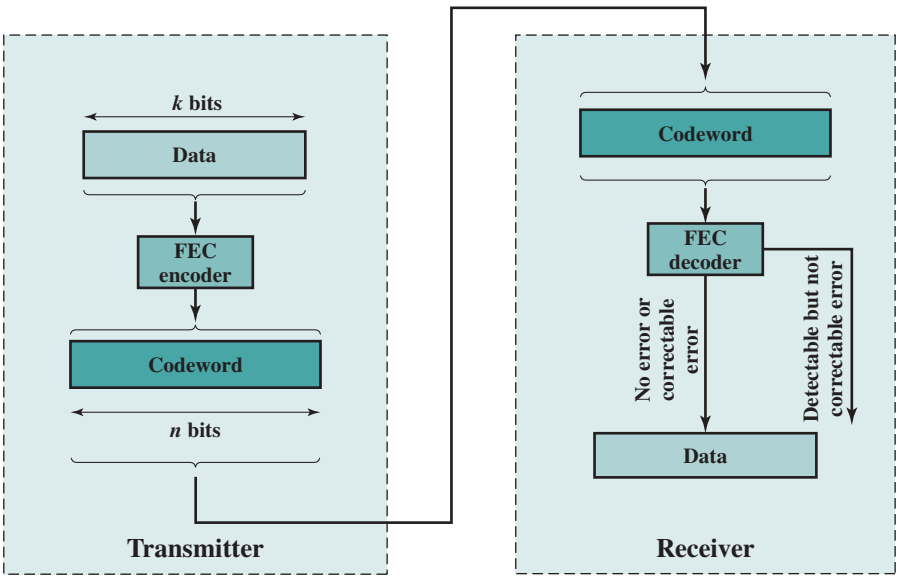


Figure 6.8 Error-Correction Process

- **Detectable, correctable errors:** For certain error patterns, it is possible for the decoder to detect and correct those errors. Thus, even though the incoming data block differs from the transmitted codeword, the FEC decoder is able to map this block into the original data block.
- **Detectable, not correctable errors:** For certain error patterns, the decoder can detect but not correct the errors. In this case, the decoder simply reports an uncorrectable error.
- **Undetectable errors:** For certain, typically rare, error patterns, the decoder does not detect the error and maps the incoming n-bit data block into a k-bit block that differs from the original k-bit block.

How is it possible for the decoder to correct bit errors? In essence, error correction works by adding sufficient redundancy to the transmitted message. The redundancy makes it possible for the receiver to deduce what the original message was, even in the face of a certain level of error rate. In this section, we look at a widely used form of error-correcting code known as a block error-correcting code. Our discussion here only deals with basic principles; a discussion of specific error-correcting codes is provided in Chapter 16.

Before proceeding, we note that in many cases, the error-correcting code follows the same general layout as shown for error-detecting codes in Figure 6.2. That is, the FEC algorithm takes as input a k-bit block and adds $(n - k)$ check bits to that block to produce an n-bit block; all of the bits in the original k-bit block show up in the n-bit block. For some FEC algorithms, the FEC algorithm maps the k-bit input into an n-bit codeword in such a way that the original k bits do not appear in the codeword.

Block Code Principles

To begin, we define a term that shall be of use to us. The **Hamming distance** $d(v_1, v_2)$ between two n-bit binary sequences v_1 and v_2 is the number of bits in which v_1 and v_2 disagree. For example, if

$$v_1 = 011011, \quad v_2 = 110001$$

then

$$d(v_1, v_2) = 3$$

Now let us consider the block code technique for error correction. Suppose we wish to transmit blocks of data of length k bits. Instead of transmitting each block as k bits, we map each k-bit sequence into a unique n-bit codeword.

EXAMPLE 6.9

For $k = 2$ and $n = 5$, we can make the following assignment:

Data Block	Codeword
00	00000
01	00111
10	11001
11	11110

Now, suppose that a codeword block is received with the bit pattern 00100. This is not a valid codeword, and so the receiver has detected an error. Can the error be corrected? We cannot be sure which data block was sent because 1, 2, 3, 4, or even all 5 of the bits that were transmitted may have been corrupted by noise. However, notice that it would require only a single-bit change to transform the valid codeword 00000 into 00100. It would take two bit changes to transform 00111 to 00100, three bit changes to transform 11110 to 00100, and it would take four bit changes to transform 11001 into 00100. Thus, we can deduce that the most likely codeword that was sent was 00000 and that therefore the desired data block is 00. This is error correction. In terms of Hamming distances, we have

$$d(00000, 00100) = 1; \quad (00111, 00100) = 2;$$
$$d(11001, 00100) = 4; \quad d(11110, 00100) = 3$$

So the rule we would like to impose is that if an invalid codeword is received, then the valid codeword that is closest to it (minimum distance) is selected. This will only work if there is a unique valid codeword at a minimum distance from each invalid codeword.

For our example, it is not true that for every invalid codeword there is one and only one valid codeword at a minimum distance. There are $2^5 = 32$ possible

codewords of which 4 are valid, leaving 28 invalid codewords. For the invalid codewords, we have the following:

Invalid Codeword	Minimum Distance	Valid Codeword	Invalid Codeword	Minimum Distance	Valid Codeword
00001	1	00000	10000	1	00000
00010	1	00000	10001	1	11001
00011	1	00111	10010	2	00000 or 11110
00100	1	00000	10011	2	00111 or 11001
00101	1	00111	10100	2	00000 or 11110
00110	1	00111	10101	2	00111 or 11001
01000	1	00000	10110	1	11110
01001	1	11001	10111	1	00111
01010	2	00000 or 11110	11000	1	11001
01011	2	00111 or 11001	11010	1	11110
01100	2	00000 or 11110	11011	1	11001
01101	2	00111 or 11001	11100	1	11110
01110	1	11110	11101	1	11001
01111	1	00111	11111	1	11110

There are eight cases in which an invalid codeword is at a distance 2 from two different valid codewords. Thus, if one such invalid codeword is received, an error in 2 bits could have caused it and the receiver has no way to choose between the two alternatives. An error is detected but cannot be corrected. However, in every case in which a single-bit error occurs, the resulting codeword is of distance 1 from only one valid codeword and the decision can be made. This code is therefore capable of correcting all single-bit errors but cannot correct double-bit errors. Another way to see this is to look at the pairwise distances between valid codewords:

$$d(00000, 00111) = 3; \quad d(00000, 11001) = 3; \quad d(00000, 11110) = 4;$$
$$d(00111, 11001) = 4; \quad d(00111, 11110) = 3; \quad d(11001, 11110) = 3$$

The minimum distance between valid codewords is 3. Therefore, a single-bit error will result in an invalid codeword that is a distance 1 from the original valid codeword but a distance at least 2 from all other valid codewords. As a result, the code can always correct a single-bit error. Note that the code also will always detect a double-bit error.

The preceding example illustrates the essential properties of a block error-correcting code. An (n, k) block code encodes k data bits into n-bit codewords. Typically, each valid codeword reproduces the original k data bits and adds to them $(n - k)$ check bits to form the n-bit codeword. Thus the design of a block code is equivalent to the design of a function of the form $\mathbf{v_c} = f(\mathbf{v_d})$, where $\mathbf{v_d}$ is a vector of k data bits and $\mathbf{v_c}$ is a vector of n codeword bits.

With an (n, k) block code, there are 2^k valid codewords out of a total of 2^n possible codewords. The ratio of redundant bits to data bits, $(n - k)/k$, is called

the **redundancy** of the code, and the ratio of data bits to total bits, k/n, is called the **code rate**. The code rate is a measure of how much additional bandwidth is required to carry data at the same data rate as without the code. For example, a code rate of 1/2 requires double the transmission capacity of an uncoded system to maintain the same data rate. Our example has a code rate of 2/5 and so requires 2.5 times the capacity of an uncoded system. For example, if the data rate input to the encoder is 1 Mbps, then the output from the encoder must be at a rate of 2.5 Mbps to keep up.

For a code consisting of the codewords $\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_s$, where $s = 2^n$, the minimum distance $d_{\min}$ of the code is defined as

$$d_{\min} = \min_{i \neq j} [d(\mathbf{w}_i, \mathbf{w}_j)]$$

It can be shown that the following conditions hold. For a given positive integer t, if a code satisfies $d_{\min} \geq (2t + 1)$, then the code can correct all bit errors up to and including errors of t bits. If $d_{\min} \geq 2t$, then all errors $\leq (t - 1)$ bits can be corrected and errors of t bits can be detected but not, in general, corrected. Conversely, any code for which all errors of magnitude $\leq t$ are corrected must satisfy $d_{\min} \geq (2t + 1)$, and any code for which all errors of magnitude $\leq (t - 1)$ are corrected and all errors of magnitude t are detected must satisfy $d_{\min} \geq 2t$.

Another way of putting the relationship between $d_{\min}$ and t is to say that the maximum number of guaranteed correctable errors per codeword satisfies

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$$

where $\lfloor x \rfloor$ means the largest integer not to exceed x (e.g., $\lfloor 6.3 \rfloor = 6$). Furthermore, if we are concerned only with error detection and not error correction, then the number of errors, t, that can be detected satisfies

$$t = d_{\min} - 1$$

To see this, consider that if $d_{\min}$ errors occur, this could change one valid codeword into another. Any number of errors less than $d_{\min}$ cannot result in another valid codeword.

The design of a block code involves a number of considerations.

1. For given values of n and k, we would like the largest possible value of $d_{\min}$.
2. The code should be relatively easy to encode and decode, requiring minimal memory and processing time.
3. We would like the number of extra bits $(n - k)$ to be small, to reduce bandwidth.
4. We would like the number of extra bits $(n - k)$ to be large, to reduce error rate.

Clearly, the last two objectives are in conflict, and trade-offs must be made.

It is instructive to examine Figure 6.9. The literature on error-correcting codes frequently includes graphs of this sort to demonstrate the effectiveness of various

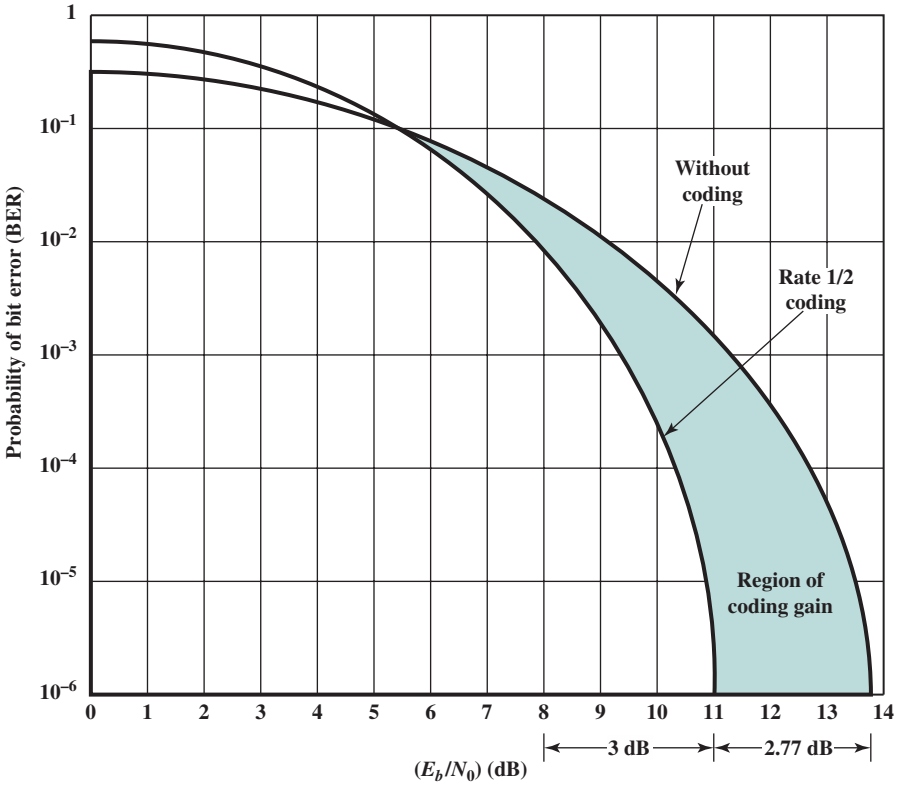


Figure 6.9 How Coding Improves System Performance

encoding schemes. Recall from Chapter 5 that coding can be used to reduce the required E_b/N_0 value to achieve a given bit error rate.[6] The coding discussed in Chapter 5 has to do with the definition of signal elements to represent bits. The coding discussed in this chapter also has an effect on E_b/N_0. In Figure 6.9, the curve on the right is for an uncoded modulation system. To the left of that curve is the area in which improvement can be achieved. In this region, a smaller BER (bit error rate) is achieved for a given E_b/N_0, and conversely, for a given BER, a smaller E_b/N_0 is required. The other curve is a typical result of a code rate of one-half (equal number of data and check bits). Note that at an error rate of 10^{-6}, the use of coding allows a reduction in E_b/N_0 of 2.77 dB. This reduction is referred to as the **coding gain**, which is defined as the reduction, in decibels, in the required E_b/N_0 to achieve a specified BER of an error-correcting coded system compared to an uncoded system using the same modulation.

It is important to realize that the BER for the second rate 1/2 curve refers to the rate of uncorrected errors and that the E_b value refers to the energy per data bit.

[6] E_b/N_0 is the ratio of signal energy per bit to noise power density per Hz; it is defined and discussed in Chapter 3.

Because the rate is 1/2, there are two bits on the channel for each data bit, and the energy per coded bit is half that of the energy per data bit, or a reduction of 3 dB to a value of 8 dB. If we look at the energy per coded bit for this system, then we see that the channel bit error rate is about 2.4×10^{-2}, or 0.024.

Finally, note that below a certain threshold of E_b/N_0, the coding scheme actually degrades performance. In our example of Figure 6.9, the threshold occurs at about 5.4 dB. Below the threshold, the extra check bits add overhead to the system that reduces the energy per data bit causing increased errors. Above the threshold, the error-correcting power of the code more than compensates for the reduced E_b, resulting in a coding gain.

6.7 RECOMMENDED READING AND ANIMATIONS

The classic treatment of error-detecting codes and CRC is [PETE61]. [RAMA88] is an excellent tutorial on CRC.

[ADAM91] provides comprehensive treatment of error-correcting codes. [SKLA01] contains a clear, well-written section on the subject. Two useful survey articles are [BERL87] and [BHAR83]. A quite readable theoretical and mathematical treatment of error-correcting codes is [ASH90]. [LIN04] is a thorough treatment of both error-detecting and error-correction codes.

ADAM91 Adamek, J. *Foundations of Coding.* New York: Wiley, 1991.

ASH90 Ash, R. *Information Theory.* New York: Dover, 1990.

BERL87 Berlekamp, E.; Peile, R.; and Pope, S. "The Application of Error Control to Communications." *IEEE Communications Magazine,* April 1987.

BHAR83 Bhargava, V. "Forward Error Correction Schemes for Digital Communications." *IEEE Communications Magazine,* January 1983.

LIN04 Lin, S., and Costello, D. *Error Control Coding.* Upper Saddle River, NJ: Prentice Hall, 2004.

PETE61 Peterson, W., and Brown, D. "Cyclic Codes for Error Detection." *Proceedings of the IEEE,* January 1961.

RAMA88 Ramabadran, T., and Gaitonde, S. "A Tutorial on CRC Computations." *IEEE Micro,* August 1988.

SKLA01 Sklar, B. *Digital Communications: Fundamentals and Applications.* Upper Saddle River, NJ: Prentice Hall, 2001.

Animations

Animations that illustrate CRC and parity are available at the Premium Web site. The reader is encouraged to view these animations to reinforce concepts from this chapter.

6.8 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

check bits	error-correcting code (ECC)	Hamming distance
checksum	error detection	ones-complement addition
codeword	error-detecting code	ones-complement operation
code rate	forward error correction (FEC)	parity bit
coding gain	frame	parity check
cyclic redundancy check (CRC)	frame check sequence (FCS)	redundancy
error correction		two-dimensional parity check

Review Questions

6.1 What is a parity bit?
6.2 What is the CRC?
6.3 Why would you expect a CRC to detect more errors than a parity bit?
6.4 List three different ways in which the CRC algorithm can be described.
6.5 Is it possible to design an ECC that will correct some double-bit errors but not all double-bit errors? Why or why not?
6.6 In an (n, k) block ECC, what do n and k represent?

Problems

6.1 For the example of Figure 6.1, what is the minimum and maximum value of x that satisfies the formal definition of error burst in Section 6.1?
6.2 Would you expect that the inclusion of a parity bit with each character would change the probability of receiving a correct message?
6.3 Two communicating devices are using a single-bit even parity check for error detection. The transmitter sends the byte 10101010 and, because of channel noise, the receiver gets the byte 10011010. Will the receiver detect the error? Why or why not?
6.4 Consider a frame consisting of two characters of four bits each. Assume that the probability of bit error is 10^{-3} and that it is independent for each bit.
a. What is the probability that the received frame contains at least one error?
b. Now add a parity bit to each character. What is the probability?
6.5 Show that the value of p in Figure 6.3a is the same whether it is calculated as the parity of all the data bits, the parity of all the row parity bits, or the parity of all the column parity bits.
6.6 Compute the Internet checksum for the data block E3 4F 23 96 44 27 99 F3. Then perform the verification calculation.
6.7 One nice property of the Internet checksum is that it is endian-independent. Little Endian computers store hex numbers with the least significant byte last (Intel processors, for example). Big Endian computers put the least significant byte first (IBM mainframes, for example). Explain why the checksum does not need to take into account endianness.
6.8 The high-speed transport protocol XTP (Xpress Transfer Protocol) uses a 32-bit checksum function defined as the concatenation of two 16-bit functions: XOR and

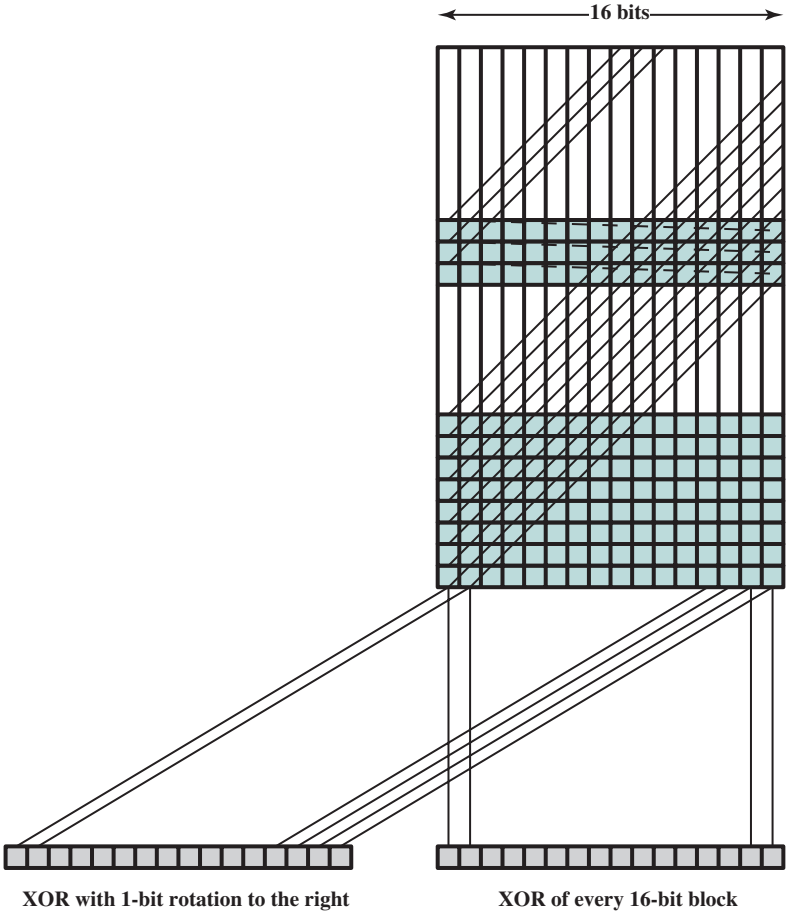


Figure 6.10 The XTP Checksum Scheme

RXOR are illustrated in Figure 6.10. The XOR function calculates the column parity. RXOR is a diagonal parity check, achieved by rotating each successive 16-bit word in the data block one bit and then performing a bitwise XOR.

a. Will this checksum detect all errors caused by an odd number of error bits? Explain.

b. Will this checksum detect all errors caused by an even number of error bits? If not, characterize the error patterns that will cause the checksum to fail.

6.9 What is the purpose of using modulo 2 arithmetic rather than binary arithmetic in computing an FCS?

6.10 Using the CRC-CCITT polynomial, generate the 16-bit CRC code for a message consisting of a 1 followed by 15 0s.

a. Use long division.

b. Use the shift-register mechanism shown in Figure 6.4.

6.11 Explain in words why the shift-register implementation of CRC will result in all 0s at the receiver if there are no errors. Demonstrate by example.

6.12 For $P = 110011$ and $M = 11100011$, find the CRC.

6.13 A CRC is constructed to generate a 4-bit FCS for an 11-bit message. The generator polynomial is $X^4 + X^3 + 1$.

a. Draw the shift-register circuit that would perform this task (see Figure 6.4).
b. Encode the data bit sequence 10011011100 (leftmost bit is the least significant) using the generator polynomial and give the codeword.
c. Now assume that bit 7 (counting from the LSB) in the codeword is in error and show that the detection algorithm detects the error.

6.14 **a.** In a CRC error-detecting scheme, choose $P(X) = X^4 + X + 1$. Encode the bits 10010011011.
b. Suppose the channel introduces an error pattern 100010000000000 (i.e., a flip from 1 to 0 or from 0 to 1 in position 1 and 5). What is received? Can the error be detected?
c. Repeat part (b) with error pattern 100110000000000.

6.15 A modified CRC procedure is commonly used in communications standards. It is defined as follows:

$$\frac{X^{16}D(X) + X^kL(X)}{P(X)} = Q + \frac{R(X)}{P(X)}$$

$$\text{FCS} = L(X) + R(X)$$

where

$$L(X) = X^{15} + X^{14} + X^{13} + \cdots + X + 1$$

and k is the number of bits being checked (address, control, and information fields).

a. Describe in words the effect of this procedure.
b. Explain the potential benefits.
c. Show a shift-register implementation for $P(X) = X^{16} + X^{12} + X^5 + 1$.

6.16 Calculate the Hamming pairwise distances among the following codewords:
a. 00000, 10101, 01010
b. 000000, 010101, 101010, 110110

6.17 Section 6.6 discusses block error-correcting codes that make a decision on the basis of minimum distance. That is, given a code consisting of s equally likely codewords of length n, for each received sequence $\mathbf{v}$, the receiver selects the codeword $\mathbf{w}$ for which the distance $d(\mathbf{w}, \mathbf{v})$ is a minimum. We would like to prove that this scheme is "ideal" in the sense that the receiver always selects the codeword for which the probability of $\mathbf{w}$ given $\mathbf{v}$, $p(\mathbf{w}|\mathbf{v})$, is a maximum. Because all codewords are assumed equally likely, the codeword that maximizes $p(\mathbf{w}|\mathbf{v})$ is the same as the codeword that maximizes $p(\mathbf{v}|\mathbf{w})$.

a. In order that $\mathbf{w}$ be received as $\mathbf{v}$, there must be exactly $d(\mathbf{w}, \mathbf{v})$ errors in transmission, and these errors must occur in those bits where $\mathbf{w}$ and $\mathbf{v}$ disagree. Let β be the probability that a given bit is transmitted incorrectly and n be the length of a codeword. Write an expression for $p(\mathbf{v}|\mathbf{w})$ as a function of β, $d(\mathbf{w}, \mathbf{v})$, and n. *Hint:* The number of bits in error is $d(\mathbf{w}, \mathbf{v})$ and the number of bits not in error is $n - d(\mathbf{w}, \mathbf{v})$.
b. Now compare $p(\mathbf{v}|\mathbf{w}_1)$ and $p(\mathbf{v}|\mathbf{w}_2)$ for two different codewords $\mathbf{w}_1$ and $\mathbf{w}_2$ by calculating $p(\mathbf{v}|\mathbf{w}_1)/p(\mathbf{v}|\mathbf{w}_2)$.
c. Assume that $0 < \beta < 0.5$ and show that $p(\mathbf{v}|\mathbf{w}_1) > p(\mathbf{v}|\mathbf{w}_2)$ if and only if $d(\mathbf{v},\mathbf{w}_1) < d(\mathbf{v},\mathbf{w}_2)$. This proves that the codeword $\mathbf{w}$ that gives the largest value of $p(\mathbf{v}|\mathbf{w})$ is that word whose distance from $\mathbf{v}$ is a minimum.

6.18 Section 6.6 states that for a given positive integer t, if a code satisfies $d_{\min} \geq 2t + 1$, then the code can correct all bit errors up to and including errors of t bits. Prove this assertion. *Hint:* Start by observing that for a codeword $\mathbf{w}$ to be decoded as another codeword $\mathbf{w}'$, the received sequence must be at least as close to $\mathbf{w}'$ as to $\mathbf{w}$.

6.19 A common technique for implementing CRC is to use a table lookup algorithm. The document site at box.com/dcc10e contains several papers describing this approach. Write a short paper that summarizes the general approach to implementing CRC using table lookup.

CHAPTER 7

DATA LINK CONTROL PROTOCOLS

LEARNING OBJECTIVES

After reading this chapter, you should be able to:

- Explain the need for a flow control and error control.
- Present an overview of the basic mechanisms of stop-and-wait flow control and sliding-window flow control.
- Present an overview of the basic mechanisms of stop-and-wait ARQ, go-back-N ARQ, and selective reject flow control.
- Present an overview of HDLC.

Our discussion so far has concerned *sending signals over a transmission link*. For effective digital data communications, much more is needed to control and manage the exchange. In this chapter, we shift our emphasis to that of *sending data over a data communications link*. To achieve the necessary control, a layer of logic is added above the physical layer discussed in Chapter 6; this logic is referred to as **data link control** or a **data link control protocol**. When a data link control protocol is used, the transmission medium between systems is referred to as a **data link**.

To see the need for data link control, we list some of the requirements and objectives for effective data communication between two directly connected transmitting-receiving stations:

- **Frame synchronization:** Data are sent in blocks called frames. The beginning and end of each frame must be recognizable. We briefly introduced this topic with the discussion of synchronous frames (Figure 6.2).
- **Flow control:** The sending station must not send frames at a rate faster than the receiving station can absorb them.
- **Error control:** Bit errors introduced by the transmission system should be corrected.
- **Addressing:** On a shared link, such as a local area network (LAN), the identity of the two stations involved in a transmission must be specified.
- **Control and data on same link:** It is usually not desirable to have a physically separate communications path for control information. Accordingly, the receiver must be able to distinguish control information from the data being transmitted.
- **Link management:** The initiation, maintenance, and termination of a sustained data exchange require a fair amount of coordination and cooperation among stations. Procedures for the management of this exchange are required.

None of these requirements is satisfied by the techniques described in Chapter 6. We shall see in this chapter that a data link protocol that satisfies these requirements is a rather complex affair. We begin by looking at two key mechanisms that are part of data link control: flow control and error control. Following this background we look at the most important example of a data link control protocol: HDLC (high-level data link control). This protocol is important for two reasons: First, it is a widely used standardized data link control protocol. Second, HDLC serves as a baseline from which virtually all other important data link control protocols are derived.

7.1 FLOW CONTROL

Flow control is a technique for assuring that a transmitting entity does not overwhelm a receiving entity with data. The receiving entity typically allocates a data buffer of some maximum length for a transfer. When data are received, the receiver must do a certain amount of processing before passing the data to the higher-level software. In the absence of flow control, the receiver's buffer may fill up and overflow while it is processing old data.

To begin, we examine mechanisms for flow control in the absence of errors. The model we will use is depicted in Figure 7.1a, which is a vertical time sequence diagram. It has the advantages of showing time dependencies and illustrating the correct send–receive relationship. Each arrow represents a single frame transiting a data link between two stations. The data are sent in a sequence of frames, with each frame containing a portion of the data and some control information. The time it takes for a station to emit all of the bits of a frame onto the medium is the **transmission time**; this is proportional to the length of the frame. The **propagation time** is the time it takes for a bit to traverse the link between source and destination. For this section, we assume that all frames that are transmitted are successfully received; no frames are lost and none arrive with errors. Furthermore, frames arrive in the same order in which they are sent. However, each transmitted frame suffers an arbitrary and variable amount of delay before reception.[1]

Stop-and-Wait Flow Control

The simplest form of flow control, known as stop-and-wait flow control, works as follows. A source entity transmits a frame. After the destination entity receives the frame, it indicates its willingness to accept another frame by sending back an acknowledgment to the frame just received. The source must wait until it receives the acknowledgment before sending the next frame. The destination can thus stop

[1]On a direct point-to-point link, the amount of delay is fixed rather than variable. However, a data link control protocol can be used over a network connection, such as a circuit-switched or ATM network, in which case the delay may be variable.

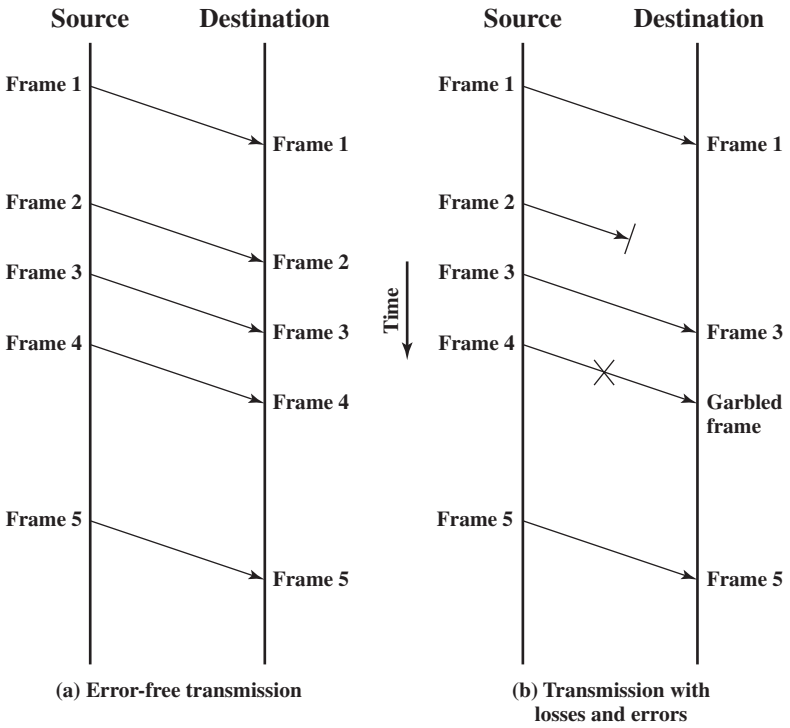


Figure 7.1 Model of Frame Transmission

the flow of data simply by withholding acknowledgment. This procedure works fine and, indeed, can hardly be improved upon when a message is sent in a few large frames. However, it is often the case that a source will break up a large block of data into smaller blocks and transmit the data in many frames. This is done for the following reasons:

- The buffer size of the receiver may be limited.
- The longer the transmission, the more likely that there will be an error, necessitating retransmission of the entire frame. With smaller frames, errors are detected sooner, and a smaller amount of data needs to be retransmitted.
- On a shared medium, such as a LAN, it is usually desirable not to permit one station to occupy the medium for an extended period, thus causing long delays at the other sending stations.

With the use of multiple frames for a single message, the stop-and-wait procedure may be inadequate. The essence of the problem is that only one frame at a time can be in transit. To explain we first define the **bit length of a link** as follows:

$$B = R \times \frac{d}{V} \tag{7.1}$$

where

B = length of the link in bits; this is the number of bits present on the link at an instance in time when a stream of bits fully occupies the link

R = data rate of the link, in bps

d = length, or distance, of the link in meters

V = velocity of propagation, in m/s

In situations where the bit length of the link is greater than the frame length, serious inefficiencies result. This is illustrated in Figure 7.2. In the figure, the transmission time is normalized to one, and the propagation delay is expressed as the variable a. Thus, we can express a as

$$a = \frac{B}{L} \tag{7.2}$$

where L is the number of bits in the frame (length of the frame in bits).

When a is less than 1, the propagation time is less than the transmission time. In this case, the frame is sufficiently long that the first bits of the frame have arrived

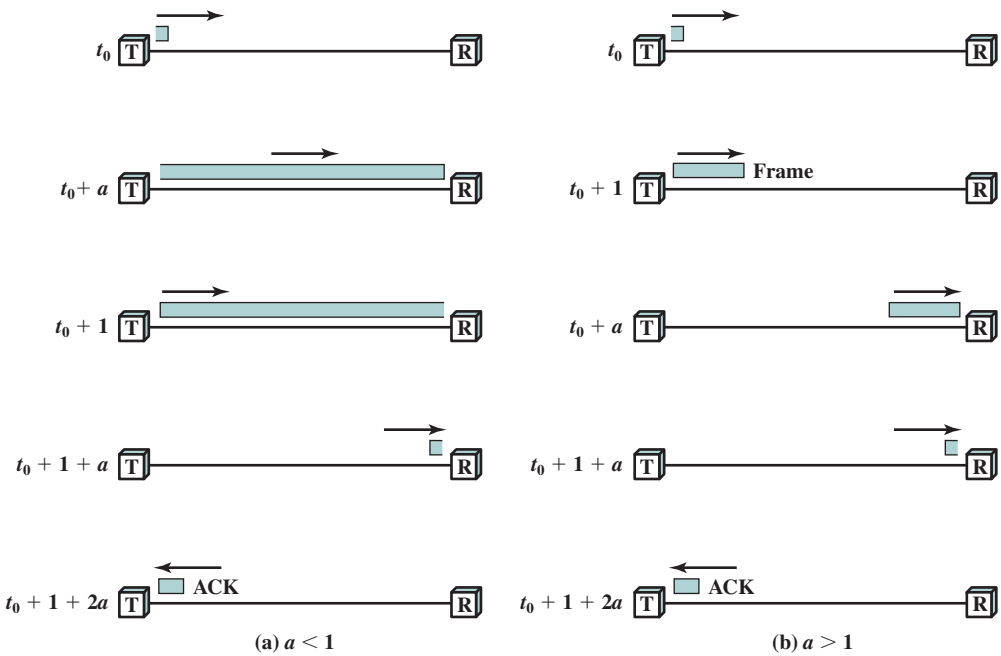


Figure 7.2 Stop-and-Wait Link Utilization (transmission time = 1; propagation time = a)

at the destination before the source has completed the transmission of the frame. When a is greater than 1, the propagation time is greater than the transmission time. In this case, the sender completes transmission of the entire frame before the leading bits of that frame arrive at the receiver. Put another way, larger values of a are consistent with higher data rates and/or longer distances between stations. Chapter 16 discusses a and data link performance.

Both parts of Figure 7.2 (a and b) consist of a sequence of snapshots of the transmission process over time. In both cases, the first four snapshots show the process of transmitting a frame containing data, and the last snapshot shows the return of a small acknowledgment frame. Note that for $a > 1$, the line is always underutilized and even for $a < 1$, the line is inefficiently utilized. In essence, for very high data rates, for very long distances between sender and receiver, stop-and-wait flow control provides inefficient line utilization.

EXAMPLE 7.1 Consider a 200-m optical fiber link operating at 1 Gbps. The velocity of propagation of optical fiber is typically about 2×10^8 m/s. Using Equation (7.1), $B = (10^9 \times 200)/(2 \times 10^8) = 1000$ bits. Assume a frame of 1000 octets, or 8000 bits, is transmitted. Using Equation (7.2), $a = (1000/8000) = 0.125$. Using Figure 7.2a as a guide, assume transmission starts at time $t = 0$. After 1 μs (a normalized time of 0.125 frame times), the leading edge (first bit) of the frame has reached R, and the first 1000 bits of the frame are spread out across the link. At time $t = 8\ \mu$s, the trailing edge (final bit) of the frame has just been emitted by T, and the final 1000 bits of the frame are spread out across the link. At $t = 9\ \mu$s, the final bit of the frame arrives at R. R now sends back an ACK frame. If we assume the frame transmission time is negligible (very small ACK frame) and that the ACK is sent immediately, the ACK arrives at T at $t = 10\ \mu$s. At this point, T can begin transmitting a new frame. The actual transmission time for the frame was 8 μs, but the total time to transmit the first frame and receive an ACK is 10 μs.

Now consider a 1-Mbps link between two ground stations that communicate via a satellite relay. A geosynchronous satellite has an altitude of roughly 36,000 km. Then $B = (10^6 \times 2 \times 36{,}000{,}000)/(3 \times 10^8) = 240{,}000$ bits. For a frame length of 8000 bits, $a = (240{,}000/8000) = 30$. Using Figure 7.2b as a guide, we can work through the same steps as before. In this case, it takes 240 ms for the leading edge of the frame to arrive and an additional 8 ms for the entire frame to arrive. The ACK arrives back at T at $t = 488$ ms. The actual transmission time for the first frame was 8 ms, but the total time to transmit the first frame and receive an ACK is 488 ms.

Sliding-Window Flow Control

The essence of the problem described so far is that only one frame at a time can be in transit. In situations where the bit length of the link is greater than the frame length ($a > 1$), serious inefficiencies result. Efficiency can be greatly improved by allowing multiple frames to be in transit at the same time.

Let us examine how this might work for two stations, A and B, connected via a full-duplex link. Station B allocates buffer space for W frames. Thus, B can accept W frames, and A is allowed to send W frames without waiting for any acknowledgments. To keep track of which frames have been acknowledged, each is labeled with a sequence number. B acknowledges a frame by sending an acknowledgment that includes the sequence number of the next frame expected. This acknowledgment also implicitly announces that B is prepared to receive the next W frames, beginning with the number specified. This scheme can also be used to acknowledge multiple frames. For example, B could receive frames 2, 3, and 4 but withhold acknowledgment until frame 4 has arrived. By then returning an acknowledgment with sequence number 5, B acknowledges frames 2, 3, and 4 at one time. A maintains a list of sequence numbers that it is allowed to send, and B maintains a list of sequence numbers that it is prepared to receive. Each of these lists can be thought of as a *window* of frames. The operation is referred to as **sliding-window flow control**.

Several additional comments need to be made. Because the sequence number to be used occupies a field in the frame, it is limited to a range of values. For example, for a 3-bit field, the sequence number can range from 0 to 7. Accordingly, frames are numbered modulo 8; that is, after sequence number 7, the next number is 0. In general, for a k-bit field the range of sequence numbers is 0 through $2^k - 1$, and frames are numbered modulo 2^k. As will be shown subsequently, the maximum window size is $2^k - 1$.

Figure 7.3 is a useful way of depicting the sliding-window process. It assumes the use of a 3-bit sequence number, so that frames are numbered sequentially from

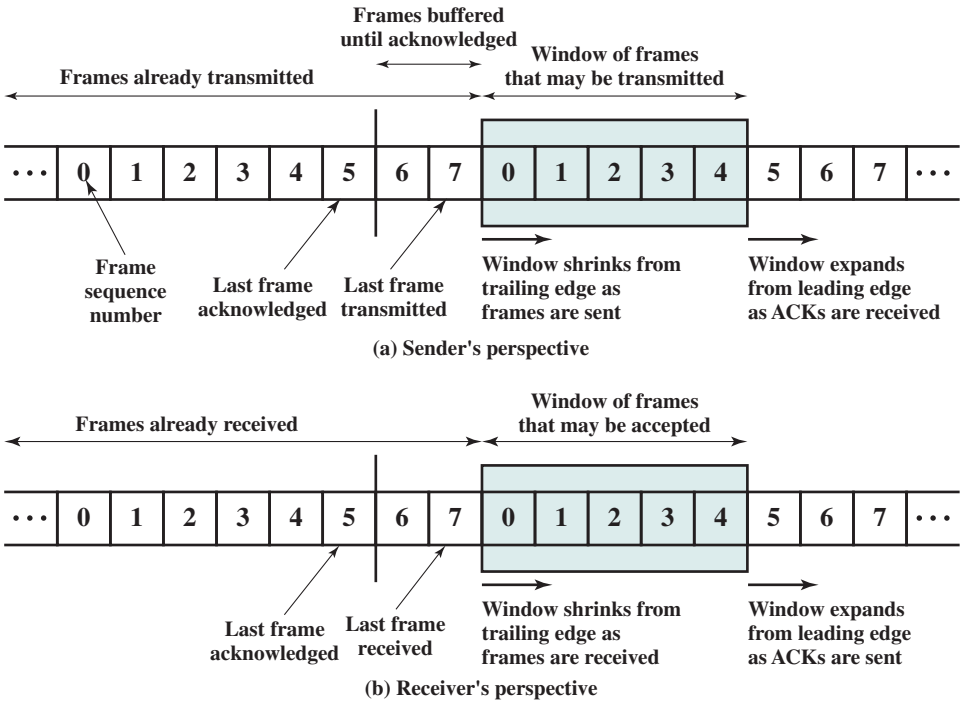


Figure 7.3 Sliding-Window Depiction

0 through 7, and then the same numbers are reused for subsequent frames. The shaded rectangle indicates the frames that may be sent; in this figure, the sender may transmit five frames, beginning with frame 0. Each time a frame is sent, the shaded window shrinks; each time an acknowledgment is received, the shaded window grows. Frames between the vertical bar and the shaded window have been sent but not yet acknowledged. As we shall see, the sender must buffer these frames in case they need to be retransmitted.

The window size need not be the maximum possible size for a given sequence number length. For example, using a 3-bit sequence number, a window size of 5 could be configured for the stations using the sliding-window flow control protocol.

EXAMPLE 7.2 An example is shown in Figure 7.4. The example assumes a 3-bit sequence number field and a maximum window size of seven frames. Initially, A and B have windows indicating that A may transmit seven frames, beginning with frame 0 (F0). After transmitting three frames (F0, F1, F2) without acknowledgment, A has shrunk its window to four frames and maintains a copy of the three transmitted frames. The window indicates that A may transmit four frames, beginning with frame number 3. B then transmits an RR (receive ready) 3, which means "I have received all frames up through frame number 2 and am ready to receive frame number 3; in fact, I am prepared to receive seven frames, beginning with frame number 3." With this acknowledgment, A is back up to permission to transmit seven frames, still beginning with frame 3; also A may discard the buffered frames that have now been acknowledged. A proceeds to transmit frames 3, 4, 5, and 6. B returns RR 4, which acknowledges F3, and allows transmission of F4 through the next instance of F2. By the time this RR reaches A, it has already transmitted F4, F5, and F6, and therefore A may only open its window to permit sending four frames beginning with F7.

The mechanism so far described provides a form of flow control: The receiver must only be able to accommodate seven frames beyond the one it has last acknowledged. Most data link control protocols also allow a station to cut off the flow of frames from the other side by sending a Receive Not Ready (RNR) message, which acknowledges former frames but forbids transfer of future frames. Thus, RNR 5 means, "I have received all frames up through number 4 but am unable to accept any more at this time." At some subsequent point, the station must send a normal acknowledgment to reopen the window.

So far, we have discussed transmission in one direction only. If two stations exchange data, each needs to maintain two windows, one for transmit and one for receive, and each side needs to send the data and acknowledgments to the other. To provide efficient support for this requirement, a feature known as **piggybacking** is typically provided. Each **data frame** includes a field that holds the sequence number of that frame plus a field that holds the sequence number used for acknowledgment.

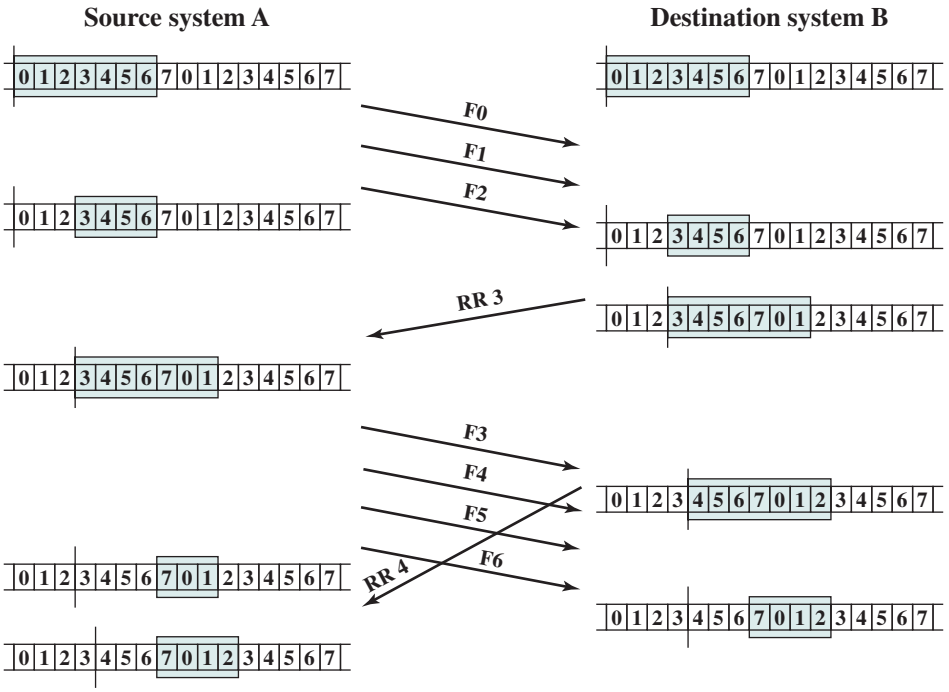


Figure 7.4 Example of a Sliding-Window Protocol

Thus, if a station has data to send and an acknowledgment to send, it sends both together in one frame, saving communication capacity. Of course, if a station has an acknowledgment but no data to send, it sends a separate **acknowledgment frame,** such as RR or RNR. If a station has data to send but no new acknowledgment to send, it must repeat the last acknowledgment sequence number that it sent. This is because the data frame includes a field for the acknowledgment number, and some value must be put into that field. When a station receives a duplicate acknowledgment, it simply ignores it.

Sliding-window flow control is potentially much more efficient than stop-and-wait flow control. The reason is that, with sliding-window flow control, the transmission link is treated as a pipeline that may be filled with frames in transit. By contrast, with stop-and-wait flow control, only one frame may be in the pipe at a time. Chapter 16 quantifies the improvement in efficiency.

EXAMPLE 7.3 Let us consider the use of sliding-window flow control for the two configurations of Example 7.1. As was calculated in Example 7.1, it takes 10 μs for an ACK to the first frame to be received. It takes 8 μs to transmit one frame, so the sender can transmit one frame and part of a second frame by the time the ACK to the first frame is received. Thus, a window size of 2 is adequate to enable the sender to transmit frames continuously, or at a rate of one frame every 8 μs. With stop-and-wait, a rate of only one frame per 10 μs is possible.

For the satellite configuration, it takes 488 ms for an ACK to the first frame to be received. It takes 8 ms to transmit one frame, so the sender can transmit 61 frames by the time the ACK to the first frame is received. With a window field of 6 bits or more, the sender can transmit continuously, or at a rate of one frame every 8 ms. If the window size is 7, using a 3-bit window field, then the sender can only send 7 frames and then must wait for an ACK before sending more. In this case, the sender can transmit at a rate of 7 frames per 488 ms, or about one frame every 70 ms. With stop-and-wait, a rate of only one frame per 488 ms is possible.

7.2 ERROR CONTROL

Error control refers to mechanisms to detect and correct errors that occur in the transmission of frames. The model that we will use, which covers the typical case, is illustrated in Figure 7.1b. As before, data are sent as a sequence of frames; frames arrive in the same order in which they are sent; and each transmitted frame suffers an arbitrary and potentially variable amount of delay before reception. In addition, we admit the possibility of two types of errors:

- **Lost frame:** A frame fails to arrive at the other side. In the case of a network, the network may simply fail to deliver a frame. In the case of a direct point-to-point data link, a noise burst may damage a frame to the extent that the receiver is not aware that a frame has been transmitted.
- **Damaged frame:** A recognizable frame does arrive, but some of the bits are in error (have been altered during transmission).

The most common techniques for error control are based on some or all of the following ingredients:

- **Error detection:** The destination detects frames that are in error, using the techniques described in the preceding chapter, and discards those frames.
- **Positive acknowledgment:** The destination returns a positive acknowledgment to successfully received, error-free frames.
- **Retransmission after timeout:** The source retransmits a frame that has not been acknowledged after a predetermined amount of time.
- **Negative acknowledgment and retransmission:** The destination returns a negative acknowledgment to frames in which an error is detected. The source retransmits such frames.

Collectively, these mechanisms are all referred to as **automatic repeat request (ARQ)**. The effect of ARQ is to turn a potentially unreliable data link into a reliable one. Three versions of ARQ have been standardized:

- Stop-and-wait ARQ
- Go-back-N ARQ
- Selective-reject ARQ

All of these forms are based on the use of the flow control techniques discussed in Section 7.1. We examine each in turn.

Stop-and-Wait ARQ

Stop-and-wait ARQ is based on the stop-and-wait flow control technique outlined previously. The source station transmits a single frame and then must await an acknowledgment (ACK). No other data frames can be sent until the destination station's reply arrives at the source station.

Two sorts of errors could occur. First, the frame that arrives at the destination could be damaged. The receiver detects this by using the error-detection technique referred to earlier and simply discards the frame. To account for this possibility, the source station is equipped with a timer. After a frame is transmitted, the source station waits for an acknowledgment. If no acknowledgment is received by the time that the timer expires, then the same frame is sent again. Note that this method requires that the transmitter maintain a copy of a transmitted frame until an acknowledgment is received for that frame.

The second sort of error is a damaged acknowledgment. Consider the following situation. Station A sends a frame. The frame is received correctly by station B, which responds with an acknowledgment. The ACK is damaged in transit and is not recognizable by A, which will therefore time out and resend the same frame. This duplicate frame arrives and is accepted by B. B has therefore accepted two copies of the same frame as if they were separate. To avoid this problem, frames are alternately labeled with 0 or 1, and positive acknowledgments are of the form ACK0 and ACK1. In keeping with the sliding-window convention, an ACK0 acknowledges receipt of a frame numbered 1 and indicates that the receiver is ready for a frame numbered 0.

Figure 7.5 gives an example of the use of stop-and-wait ARQ, showing the transmission of a sequence of frames from source A to destination B.[2] The figure shows the two types of errors just described. The third frame transmitted by A is lost or damaged and therefore B does not return an ACK. A times out and retransmits the frame. Later, A transmits a frame labeled 1 but the ACK0 for that frame is lost. A times out and retransmits the same frame. When B receives two frames in a row with the same label, it discards the second frame but sends back an ACK0 to each.

The principal advantage of stop-and-wait ARQ is its simplicity. Its principal disadvantage, as discussed in Section 7.1, is that stop-and-wait is an inefficient mechanism. The sliding-window flow control technique can be adapted to provide more efficient line use; in this context, it is sometimes referred to as *continuous ARQ*.

Go-Back-N ARQ

The form of error control based on sliding-window flow control that is most commonly used is called go-back-N ARQ. In this method, a station may send a series of frames sequentially numbered modulo some maximum value. The number of unacknowledged frames outstanding is determined by window size, using the

[2]This figure indicates the time required to transmit a frame. For simplicity, other figures in this chapter do not show this time.

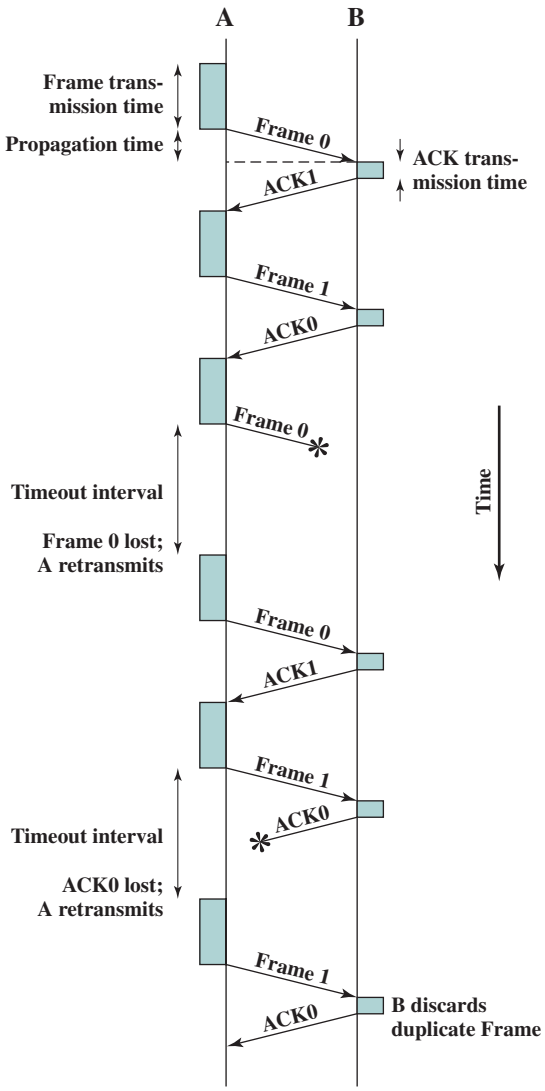


Figure 7.5 Stop-and-Wait ARQ

sliding-window flow control technique. While no errors occur, the destination will acknowledge incoming frames as usual (RR = receive ready, or piggybacked acknowledgment). If the destination station detects an error in a frame, it may send a negative acknowledgment (REJ = reject) for that frame, as explained in the following rules. The destination station will discard that frame and all future incoming frames until the frame in error is correctly received. Thus, the source station, when it receives a REJ, must retransmit the frame in error plus all succeeding frames that were transmitted in the interim.

Suppose that station A is sending frames to station B. After each transmission, A sets an acknowledgment timer for the frame just transmitted. Suppose that B has

previously successfully received frame $(i - 1)$ and A has just transmitted frame i. The go-back-N technique takes into account the following contingencies:

1. **Damaged frame.** If the received frame is invalid (i.e., B detects an error, or the frame is so damaged that B does not even perceive that it has received a frame), B discards the frame and takes no further action as the result of that frame. There are two subcases:
   a. Within a reasonable period of time, A subsequently sends frame $(i + 1)$. B receives frame $(i + 1)$ out of order and sends a REJ *i*. A must retransmit frame *i* and all subsequent frames.
   b. A does not soon send additional frames. B receives nothing and returns neither an RR nor a REJ. When A's timer expires, it transmits an RR frame that includes a bit known as the P bit, which is set to 1. B interprets the RR frame with a P bit of 1 as a command that must be acknowledged by sending an RR indicating the next frame that it expects, which is frame *i*. When A receives the RR, it retransmits frame *i*. Alternatively, A could just retransmit frame *i* when its timer expires.
2. **Damaged RR.** There are two subcases:
   a. B receives frame *i* and sends RR $(i + 1)$, which suffers an error in transit. Because acknowledgments are cumulative (e.g., RR 6 means that all frames through 5 are acknowledged), it may be that A will receive a subsequent RR to a subsequent frame and that it will arrive before the timer associated with frame *i* expires.
   b. If A's timer expires, it transmits an RR command as in Case 1b. It sets another timer, called the P-bit timer. If B fails to respond to the RR command, or if its response suffers an error in transit, then A's P-bit timer will expire. At this point, A will try again by issuing a new RR command and restarting the P-bit timer. This procedure is tried for a number of iterations. If A fails to obtain an acknowledgment after some maximum number of attempts, it initiates a reset procedure.
3. **Damaged REJ.** If a REJ is lost, this is equivalent to Case 1b.

EXAMPLE 7.4 Figure 7.6a is an example of the frame flow for go-back-N ARQ. Because of the propagation delay on the line, by the time that an acknowledgment (positive or negative) arrives back at the sending station, it has already sent at least one additional frame beyond the one being acknowledged. In this example, frame 4 is damaged. Frames 5 and 6 are received out of order and are discarded by B. When frame 5 arrives, B immediately sends a REJ 4. When the REJ to frame 4 is received, not only frame 4 but frames 5 and 6 must be retransmitted. Note that the transmitter must keep a copy of all unacknowledged frames. Figure 7.6a also shows an example of retransmission after timeout. No acknowledgment is received for frame 5 within the timeout period, so A issues an RR to determine the status of B.

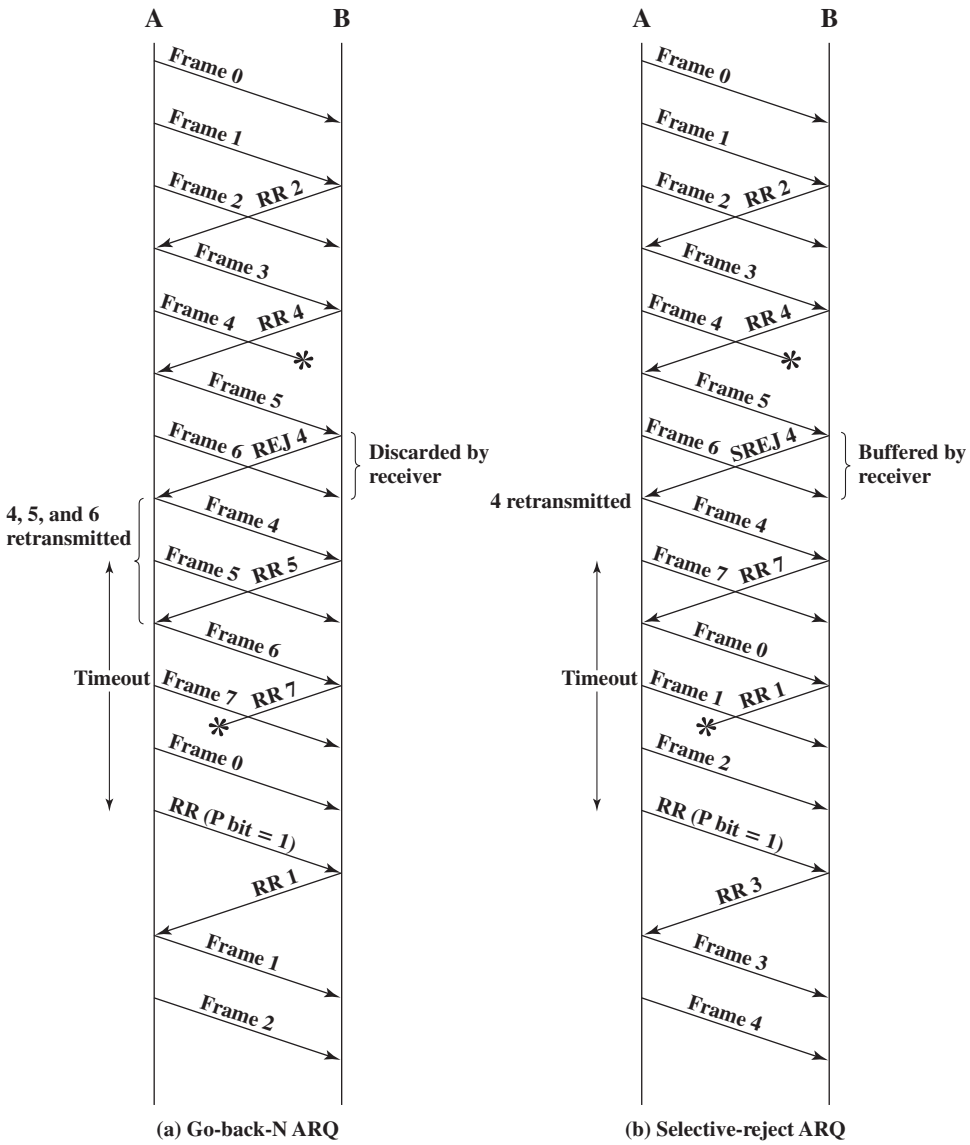


Figure 7.6 Sliding-Window ARQ Protocols

In Section 7.1, we mentioned that for a k-bit sequence number field, which provides a sequence number range of 2^k, the maximum window size is limited to $2^k - 1$. This has to do with the interaction between error control and acknowledgment. Consider that if data are being exchanged in both directions, station B must send piggybacked acknowledgments to station A's frames in the data frames being transmitted by B, even if the acknowledgment has already been sent. As we have mentioned, this is because B must put some number in the acknowledgment field of its data frame. As an example, assume a 3-bit sequence number (sequence number

space = 8). Suppose a station sends frame 0 and gets back an RR 1 and then sends frames 1, 2, 3, 4, 5, 6, 7, 0 and gets another RR 1. This could mean that all eight frames were received correctly and the RR 1 is a cumulative acknowledgment. It could also mean that all eight frames were damaged or lost in transit, and the receiving station is repeating its previous RR 1. The problem is avoided if the maximum window size is limited to 7 $(2^3 - 1)$.

Selective-Reject ARQ

With **selective-reject ARQ**, the only frames retransmitted are those that receive a negative acknowledgment, in this case called SREJ, or those that time out.

EXAMPLE 7.5 Figure 7.6b illustrates this scheme. When frame 5 is received out of order, B sends a SREJ 4, indicating that frame 4 has not been received. However, B continues to accept incoming frames and buffers them until a valid frame 4 is received. At that point, B can place the frames in the proper order for delivery to higher-layer software.

Selective reject would appear to be more efficient than go-back-N, because it minimizes the amount of retransmission. On the other hand, the receiver must maintain a buffer large enough to save post-SREJ frames until the frame in error is retransmitted and must contain logic for reinserting that frame in the proper sequence. The transmitter, too, requires more complex logic to be able to send a frame out of sequence. Because of such complications, selective-reject ARQ is much less widely used than go-back-N ARQ. Selective reject is a useful choice for a satellite link because of the long propagation delay involved.

The window size limitation is more restrictive for selective-reject than for go-back-N. Consider the case of a 3-bit sequence number size for selective-reject. Allow a window size of seven, and consider the following scenario:

1. Station A sends frames 0 through 6 to station B.
2. Station B receives all seven frames and cumulatively acknowledges with RR 7.
3. Because of a noise burst, the RR 7 is lost.
4. A times out and retransmits frame 0.
5. B has already advanced its receive window to accept frames 7,0,1,2,3,4, and 5. Thus it assumes that frame 7 has been lost and that this is a new frame 0, which it accepts.

The problem with the preceding scenario is that there is an overlap between the sending and receiving windows. To overcome the problem, the maximum window size should be no more than half the range of sequence numbers. In the preceding scenario, if only four unacknowledged frames may be outstanding, no confusion can result. In general, for a k-bit sequence number field, which provides a sequence number range of 2^k, the maximum window size is limited to 2^{k-1}.

7.3 HIGH-LEVEL DATA LINK CONTROL (HDLC)

The most important data link control protocol is HDLC (ISO 3009, ISO 4335). Not only is HDLC widely used, but it is the basis for many other important data link control protocols, which use the same or similar formats and the same mechanisms as employed in HDLC.

Basic Characteristics

To satisfy a variety of applications, HDLC defines three types of stations, two link configurations, and three data transfer modes of operation. The three station types are:

- **Primary station:** Responsible for controlling the operation of the link. Frames issued by the primary are called commands.
- **Secondary station:** Operates under the control of the primary station. Frames issued by a secondary are called responses. The primary maintains a separate logical link with each secondary station on the line.
- **Combined station:** Combines the features of primary and secondary. A combined station may issue both commands and responses.

The two link configurations are

- **Unbalanced configuration:** Consists of one primary and one or more secondary stations and supports both full-duplex and half-duplex transmission.
- **Balanced configuration:** Consists of two combined stations and supports both full-duplex and half-duplex transmission.

The three data transfer modes are

- **Normal response mode (NRM):** Used with an unbalanced configuration. The primary may initiate data transfer to a secondary, but a secondary may only transmit data in response to a command from the primary.
- **Asynchronous balanced mode (ABM):** Used with a balanced configuration. Either combined station may initiate transmission without receiving permission from the other combined station.
- **Asynchronous response mode (ARM):** Used with an unbalanced configuration. The secondary may initiate transmission without explicit permission of the primary. The primary still retains responsibility for the line, including initialization, error recovery, and logical disconnection.

NRM is used on multidrop lines, in which a number of terminals are connected to a host computer. The computer polls each terminal for input. NRM is also sometimes used on point-to-point links, particularly if the link connects a terminal or other peripheral to a computer. ABM is the most widely used of the three modes; it makes more efficient use of a full-duplex point-to-point link because there is no polling overhead. ARM is rarely used; it is applicable to some special situations in which a secondary may need to initiate transmission.

Frame Structure

HDLC uses synchronous transmission. All transmissions are in the form of frames, and a single frame format suffices for all types of data and control exchanges.

Figure 7.7 depicts the structure of the HDLC frame. The flag, address, and control fields that precede the information field are known as a **header**. The frame check sequence and flag fields following the data field are referred to as a **trailer**.

FLAG FIELDS **Flag fields** delimit the frame at both ends with the unique pattern 01111110. A single flag may be used as the closing flag for one frame and the opening flag for the next. On both sides of the user-network interface, receivers are continuously hunting for the flag sequence to synchronize on the start of a frame. While receiving a frame, a station continues to hunt for that sequence to determine the end of the frame. Because the protocol allows the presence of arbitrary bit patterns (i.e., there are no restrictions on the content of the various fields imposed by the link protocol), there is no assurance that the pattern 01111110 will not appear somewhere inside the frame, thus destroying synchronization. To avoid this problem, a procedure known as **bit stuffing** is used. For all bits between the starting and ending

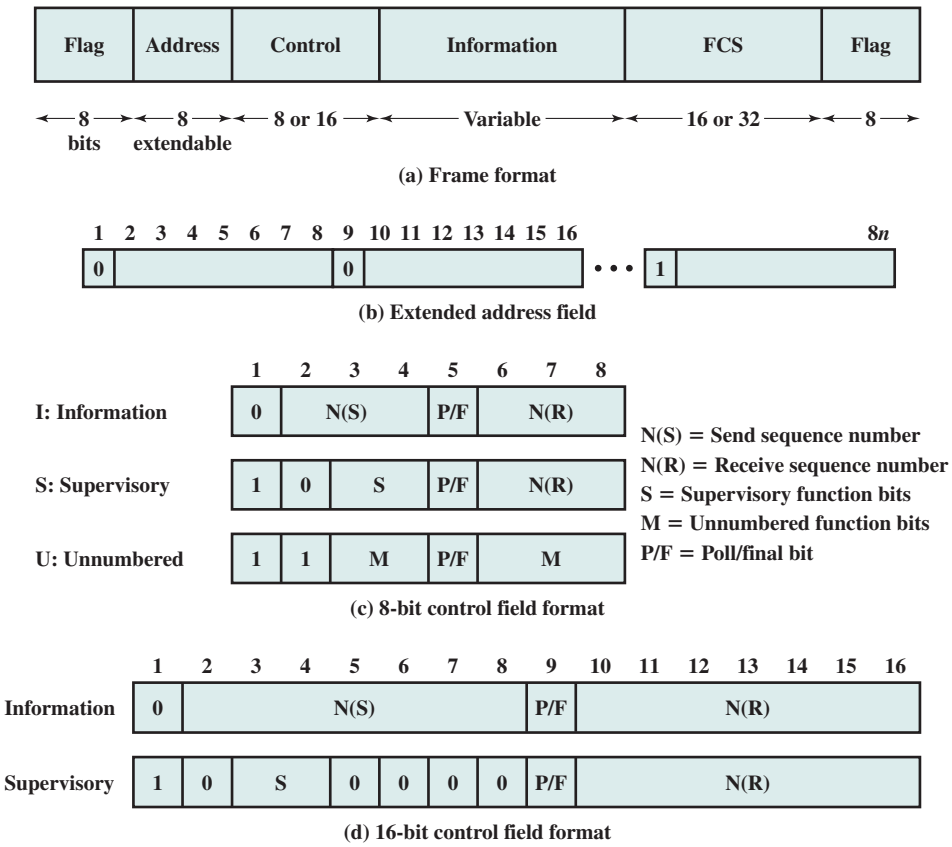


Figure 7.7 HDLC Frame Structure

Original pattern:

111111111111011111101111110

After bit-stuffing:

111110111110110111110101111010

Figure 7.8 Bit Stuffing

flags, the transmitter inserts an extra 0 bit after each occurrence of five 1s in the frame. After detecting a starting flag, the receiver monitors the bit stream. When a pattern of five 1s appears, the sixth bit is examined. If this bit is 0, it is deleted. If the sixth bit is a 1 and the seventh bit is a 0, the combination is accepted as a flag. If the sixth and seventh bits are both 1, the sender is indicating an abort condition.

With the use of bit stuffing, arbitrary bit patterns can be inserted into the data field of the frame. This property is known as **data transparency**.

Figure 7.8 shows an example of bit stuffing. Note that in the first two cases, the extra 0 is not strictly necessary for avoiding a flag pattern but is necessary for the operation of the algorithm.

ADDRESS FIELD The address field identifies the secondary station that transmitted or is to receive the frame. This field is not needed for point-to-point links but is always included for the sake of uniformity. The address field is usually 8 bits long but, by prior agreement, an extended format may be used in which the actual address length is a multiple of 7 bits. The leftmost bit of each octet is 1 or 0 according as it is or is not the last octet of the address field. The remaining 7 bits of each octet form part of the address. The single-octet address of 11111111 is interpreted as the all-stations address in both basic and extended formats. It is used to allow the primary to broadcast a frame for reception by all secondaries.

CONTROL FIELD HDLC defines three types of frames, each with a different control field format. **Information frames** (I-frames) carry the data to be transmitted for the user (the logic above HDLC that is using HDLC). Additionally, flow and error control data, using the ARQ mechanism, are piggybacked on an information frame. **Supervisory frames** (S-frames) provide the ARQ mechanism when piggybacking is not used. **Unnumbered frames** (U-frames) provide supplemental link control functions. The first one or two bits of the control field serves to identify the frame type. The remaining bit positions are organized into subfields as indicated in Figures 7.7c and d. Their use is explained in the discussion of HDLC operation later in this chapter.

All of the control field formats contain the poll/final (P/F) bit. Its use depends on context. Typically, in command frames, it is referred to as the P bit and is set to 1 to solicit (poll) a response frame from the peer HDLC entity. In response frames, it is referred to as the F bit and is set to 1 to indicate the response frame transmitted as a result of a soliciting command.

Note that the basic control field for S- and I-frames uses 3-bit sequence numbers. With the appropriate set-mode command, an extended control field can be used for S- and I-frames that employs 7-bit sequence numbers. U-frames always contain an 8-bit control field.

Information Field The information field is present only in I-frames and some U-frames. The field can contain any sequence of bits but must consist of an integral number of octets. The length of the information field is variable up to some system-defined maximum.

Frame Check Sequence Field The frame check sequence (FCS) is an error-detecting code calculated from the remaining bits of the frame, exclusive of flags. The normal code is the 16-bit CRC-CCITT defined in Section 6.3. An optional 32-bit FCS, using CRC-32, may be employed if the frame length or the line reliability dictates this choice.

Operation

HDLC operation consists of the exchange of I-frames, S-frames, and U-frames between two stations. The various commands and responses defined for these frame types are listed in Table 7.1. In describing HDLC operation, we will discuss these three types of frames.

The operation of HDLC involves three phases. First, one side or another initializes the data link so that frames may be exchanged in an orderly fashion. During this phase, the options that are to be used are agreed upon. After initialization, the two sides exchange user data and the control information to exercise flow and error control. Finally, one of the two sides signals the termination of the operation.

Initialization Either side may request initialization by issuing one of the six set-mode commands. This command serves three purposes:

1. It signals the other side that initialization is requested.
2. It specifies which of the three modes (NRM, ABM, ARM) is requested.
3. It specifies whether 3- or 7-bit sequence numbers are to be used.

If the other side accepts this request, then the HDLC module on that end transmits an unnumbered acknowledged (UA) frame back to the initiating side. If the request is rejected, then a disconnected mode (DM) frame is sent.

Data Transfer When the initialization has been requested and accepted, then a logical connection is established. Both sides may begin to send user data in I-frames, starting with sequence number 0. The N(S) and N(R) fields of the I-frame are sequence numbers that support flow control and error control. An HDLC module sending a sequence of I-frames will number them sequentially, modulo 8 or 128, depending on whether 3- or 7-bit sequence numbers are used, and place the sequence number in N(S). N(R) is the acknowledgment for I-frames received; it enables the HDLC module to indicate which number I-frame it expects to receive next.

S-frames are also used for flow control and error control. The receive ready (RR) frame acknowledges the last I-frame received by indicating the next I-frame expected. The RR is used when there is no reverse user data traffic (I-frames) to carry an acknowledgment. Receive not ready (RNR) acknowledges an I-frame, as with RR, but also asks the peer entity to suspend transmission of I-frames. When the

Table 7.1 HDLC Commands and Responses

Name	Command/ Response	Description
Information (I)	C/R	Exchange user data
Supervisory (S)		
Receive ready (RR)	C/R	Positive acknowledgment; ready to receive I-frame
Receive not ready (RNR)	C/R	Positive acknowledgment; not ready to receive
Reject (REJ)	C/R	Negative acknowledgment; go back N
Selective reject (SREJ)	C/R	Negative acknowledgment; selective reject
Unnumbered (U)		
Set normal response/extended mode (SNRM/SNRME)	C	Set mode; extended = 7-bit sequence numbers
Set asynchronous response/ extended mode (SARM/SARME)	C	Set mode; extended = 7-bit sequence numbers
Set asynchronous balanced/ extended mode (SABM, SABME)	C	Set mode; extended = 7-bit sequence numbers
Set initialization mode (SIM)	C	Initialize link control functions in addressed station
Disconnect (DISC)	C	Terminate logical link connection
Unnumbered Acknowledgment (UA)	R	Acknowledge acceptance of one of the set-mode commands
Disconnected mode (DM)	R	Responder is in disconnected mode
Request disconnect (RD)	R	Request for DISC command
Request initialization mode (RIM)	R	Initialization needed; request for SIM command
Unnumbered information (UI)	C/R	Used to exchange control information
Unnumbered poll (UP)	C	Used to solicit control information
Reset (RSET)	C	Used for recovery; resets N(R), N(S)
Exchange identification (XID)	C/R	Used to request/report status
Test (TEST)	C/R	Exchange identical information fields for testing
Frame reject (FRMR)	R	Report receipt of unacceptable frame

entity that issued RNR is again ready, it sends an RR. REJ initiates the go-back-N ARQ. It indicates that the last I-frame received has been rejected and that retransmission of all I-frames beginning with number N(R) is required. Selective reject (SREJ) is used to request retransmission of just a single frame.

DISCONNECT Either HDLC module can initiate a disconnect, either on its own initiative if there is some sort of fault, or at the request of its higher-layer user. HDLC issues a disconnect by sending a disconnect (DISC) frame. The remote entity must accept the disconnect by replying with a UA and informing its layer 3 user that the

connection has been terminated. Any outstanding unacknowledged I-frames may be lost, and their recovery is the responsibility of higher layers.

Examples of Operation To better understand HDLC operation, several examples are presented in Figure 7.9. In the example diagrams, each arrow includes a legend that specifies the frame name, the setting of the P/F bit, and, where appropriate, the values of N(R) and N(S). The setting of the P or F bit is 1 if the designation is present and 0 if absent.

Figure 7.9a shows the frames involved in link setup and disconnect. The HDLC protocol entity for one side issues an SABM command to the other side and starts

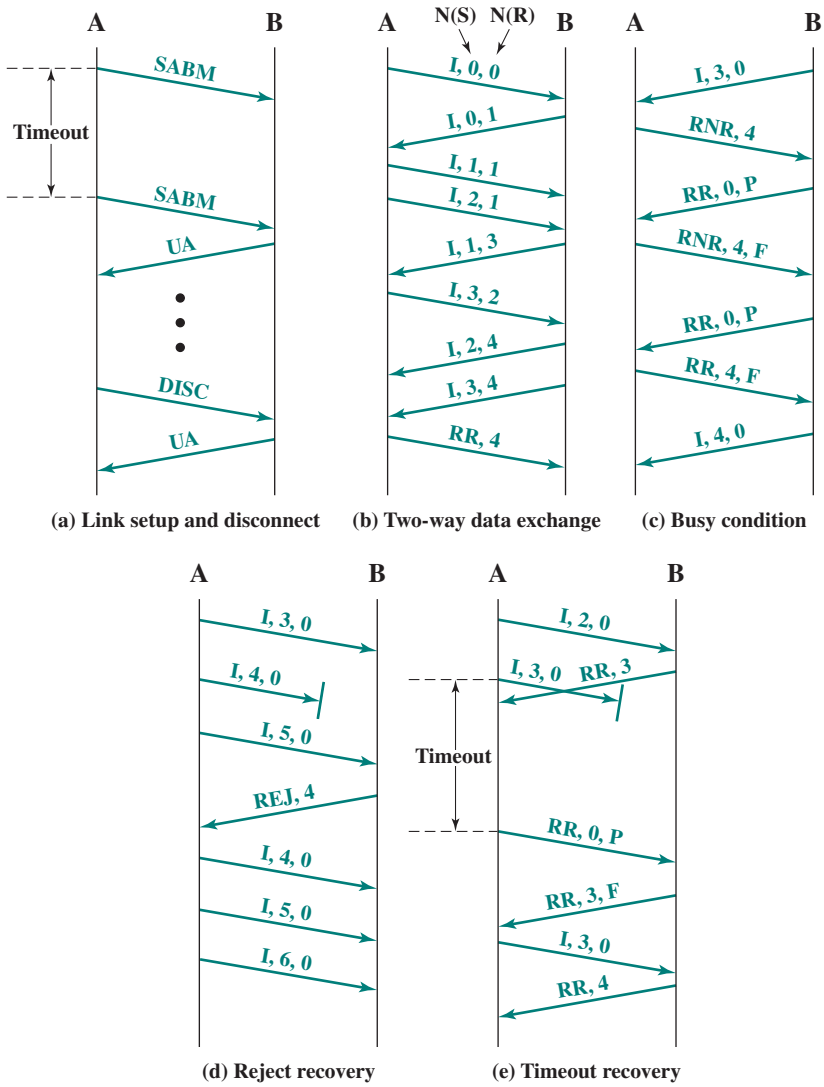


Figure 7.9 Examples of HDLC Operation

a timer. The other side, upon receiving the SABM, returns a UA response and sets local variables and counters to their initial values. The initiating entity receives the UA response, sets its variables and counters, and stops the timer. The logical connection is now active, and both sides may begin transmitting frames. Should the timer expire without a response to an SABM, the originator will repeat the SABM, as illustrated. This would be repeated until a UA or DM is received or until, after a given number of tries, the entity attempting initiation gives up and reports failure to a management entity. In such a case, higher-layer intervention is necessary. The same figure (Figure 7.9a) shows the disconnect procedure. One side issues a DISC command, and the other responds with a UA response.

Figure 7.9b illustrates the full-duplex exchange of I-frames. When an entity sends a number of I-frames in a row with no incoming data, then the receive sequence number is simply repeated (e.g., I,1,1; I,2,1 in the A-to-B direction). When an entity receives a number of I-frames in a row with no outgoing frames, then the receive sequence number in the next outgoing frame must reflect the cumulative activity (e.g., I,1,3 in the B-to-A direction). Note that, in addition to I-frames, data exchange may involve supervisory frames.

Figure 7.9c shows an operation involving a busy condition. Such a condition may arise because an HDLC entity is not able to process I-frames as fast as they are arriving, or the intended user is not able to accept data as fast as they arrive in I-frames. In either case, the entity's receive buffer fills up and it must halt the incoming flow of I-frames, using an RNR command. In this example, A issues an RNR, which requires B to halt transmission of I-frames. The station receiving the RNR will usually poll the busy station at some periodic interval by sending an RR with the P bit set. This requires the other side to respond with either an RR or an RNR. When the busy condition has cleared, A returns an RR, and I-frame transmission from B can resume.

An example of error recovery using the REJ command is shown in Figure 7.9d. In this example, A transmits I-frames numbered 3, 4, and 5. Number 4 suffers an error and is lost. When B receives I-frame number 5, it discards this frame because it is out of order and sends an REJ with an N(R) of 4. This causes A to initiate retransmission of I-frames previously sent, beginning with frame 4. A may continue to send additional frames after the retransmitted frames.

An example of error recovery using a timeout is shown in Figure 7.9e. In this example, A transmits I-frame number 3 as the last in a sequence of I-frames. The frame suffers an error. B detects the error and discards it. However, B cannot send an REJ, because there is no way to know if this was an I-frame. If an error is detected in a frame, all of the bits of that frame are suspect, and the receiver has no way to act upon it. A, however, would have started a timer as the frame was transmitted. This timer has a duration long enough to span the expected response time. When the timer expires, A initiates recovery action. This is usually done by polling the other side with an RR command with the P bit set to determine the status of the other side. Because the poll demands a response, the entity will receive a frame containing an N(R) field and be able to proceed. In this case, the response indicates that frame 3 was lost, which A retransmits.

These examples are not exhaustive. However, they should give the reader a good feel for the behavior of HDLC.

7.4 RECOMMENDED READING AND ANIMATIONS

An excellent and very detailed treatment of flow control and error control is to be found in [BERT92]. [FIOR95] points out some of the real-world reliability problems with HDLC.

BERT92 Bertsekas, D., and Gallager, R. *Data Networks*. Englewood Cliffs, NJ: Prentice Hall, 1992.

FIOR95 Fiorini, D.; Chiani, M.; Tralli, V.; and Salati, C. "Can We Trust HDLC?" *ACM Computer Communications Review*, October 1995.

Animations

An animation that illustrates data link control protocol operation is available at the Premium Web site. The reader is encouraged to view the animation to reinforce concepts from this chapter.

7.5 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

automatic repeat request (ARQ)	flag field	piggybacking
acknowledgment frame	flow control	propagation time
data frame	frame	selective-reject ARQ
data link	frame synchronization	sliding-window flow control
data link control protocol	go-back-N ARQ	stop-and-wait ARQ
data transparency	header	stop-and-wait flow control
error control	high-level data link control (HDLC)	trailer
		transmission time

Review Questions

7.1 List and briefly define some of the requirements for effective communications over a data link.

7.2 Define *flow control*.

7.3 Describe stop-and-wait flow control.

7.4 What are reasons for breaking up a long data transmission into a number of frames?

7.5 Describe sliding-window flow control.

7.6 What is the advantage of sliding-window flow control compared to stop-and-wait flow control?

7.7 What is piggybacking?
7.8 Define *error control*.
7.9 List common ingredients for error control for a link control protocol.
7.10 Describe automatic repeat request (ARQ).
7.11 List and briefly define three versions of ARQ.
7.12 What are the station types supported by HDLC? Describe each.
7.13 What are the transfer modes supported by HDLC? Describe each.
7.14 What is the purpose of the flag field?
7.15 Define *data transparency*.
7.16 What are the three frame types supported by HDLC? Describe each.

Problems

7.1 Consider a half-duplex point-to-point link using a stop-and-wait scheme, in which a series of messages is sent, with each message segmented into a number of frames. Ignore errors and frame overhead.

a. What is the effect on line utilization of increasing the message size so that fewer messages will be required? Other factors remain constant.
b. What is the effect on line utilization of increasing the number of frames for a constant message size?
c. What is the effect on line utilization of increasing frame size?

7.2 The number of bits on a transmission line that are in the process of actively being transmitted (i.e., the number of bits that have been transmitted but have not yet been received) is referred to as the *bit length* of the line. Plot the line distance versus the transmission speed for a bit length of 1000 bits. Assume a propagation velocity of 2×10^8 m/s.

7.3 In Figure 7.10 frames are generated at node A and sent to node C through node B. Determine the minimum data rate required between nodes B and C so that the buffers of node B are not flooded, based on the following:

- The data rate between A and B is 100 kbps.
- The propagation delay is 5 μs/km for both lines.
- There are full-duplex lines between the nodes.
- All data frames are 1000 bits long; ACK frames are separate frames of negligible length.
- Between A and B, a sliding-window protocol with a window size of 3 is used.
- Between B and C, stop-and-wait is used.
- There are no errors.

Hint: In order not to flood the buffers of B, the average number of frames entering and leaving B must be the same over a long interval.

7.4 A channel has a data rate of R bps and a propagation delay of t s/km. The distance between the sending and receiving nodes is L kilometers. Nodes exchange fixed-size frames of B bits. Find a formula that gives the minimum sequence field size of the frame as a function of R, t, B, and L (considering maximum utilization). Assume that ACK frames are negligible in size and the processing at the nodes is instantaneous.

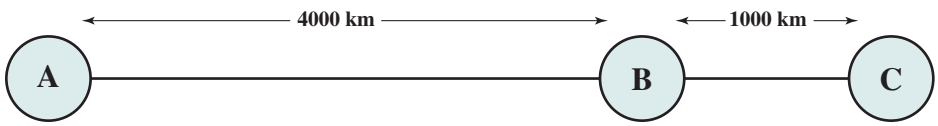


Figure 7.10 Configuration for Problem 7.3

7.5 No mention was made of reject (REJ) frames in the stop-and-wait ARQ discussion. Why is it not necessary to have REJ0 and REJ1 for stop-and-wait ARQ?

7.6 Suppose that a selective-reject ARQ is used where $W = 4$. Show, by example, that a 3-bit sequence number is needed.

7.7 Two neighboring nodes (A and B) use a sliding-window protocol with a 3-bit sequence number. As the ARQ mechanism, go-back-N is used with a window size of 4. Assuming A is transmitting and B is receiving, show the window positions for the following succession of events:

a. Before A sends any frames
b. After A sends frames 0, 1, 2 and receives acknowledgment from B for 0 and 1
c. After A sends frames 3, 4, and 5 and B acknowledges 4 and the ACK is received by A

7.8 Out-of-sequence acknowledgment cannot be used for selective-reject ARQ. That is, if frame i is rejected by station X, all subsequent I-frames and RR frames sent by X must have $N(R) = i$ until frame i is successfully received, even if other frames with $N(S) > i$ are successfully received in the meantime. One possible refinement is the following: $N(R) = j$ in an I-frame or an RR frame is interpreted to mean that frame $j - 1$ and all preceding frames are accepted except for those that have been explicitly rejected using an SREJ frame. Comment on any possible drawback to this scheme.

7.9 The ISO standard for HDLC procedures (ISO 4335) includes the following definitions: (1) an REJ condition is considered cleared upon the receipt of an incoming I-frame with an N(S) equal to the N(R) of the outgoing REJ frame; and (2) a SREJ condition is considered cleared upon the receipt of an I-frame with an N(S) equal to the N(R) of the SREJ frame. The standard includes rules concerning the relationship between REJ and SREJ frames. These rules indicate what is allowable (in terms of transmitting REJ and SREJ frames) if an REJ condition has not yet been cleared and what is allowable if an SREJ condition has not yet been cleared. Deduce the rules and justify your answer.

7.10 Two stations communicate via a 1-Mbps satellite link with a propagation delay of 270 ms. The satellite serves merely to retransmit data received from one station to another, with negligible switching delay. Using HDLC frames of 1024 bits with 3-bit sequence numbers, what is the maximum possible data throughput; that is, what is the throughput of data bits carried in HDLC frames?

7.11 It is clear that bit stuffing is needed for the address, data, and FCS fields of an HDLC frame. Is it needed for the control field?

7.12 Because of the provision that a single flag can be used as both an ending and a starting flag, a single-bit error can cause problems.

a. Explain how a single-bit error can merge two frames into one.
b. Explain how a single-bit error can split a single frame into two frames.

7.13 Suggest improvements to the bit stuffing-algorithm to overcome the problems of single-bit errors described in the preceding problem.

7.14 Using the example bit string of Figure 7.8, show the signal pattern on the line using NRZ-L coding. Does this suggest a side benefit of bit stuffing?

7.15 Assume that the primary HDLC station in NRM has sent six I-frames to a secondary. The primary's N(S) count was three (011 binary) prior to sending the six frames. If the poll bit is on in the sixth frame, what will be the N(R) count back from the secondary after the last frame? Assume error-free operation.

7.16 Consider that several physical links connect two stations. We would like to use a "multilink HDLC" that makes efficient use of these links by sending frames on a FIFO basis on the next available link. What enhancements to HDLC are needed?

7.17 A World Wide Web server is usually set up to receive relatively small messages from its clients but to transmit potentially very large messages to them. Explain, then, which type of ARQ protocol (selective reject, go-back-N) would provide less of a burden to a particularly popular WWW server.