

Error Detection and Correction

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.

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.

Types of Errors

- An error occurs when a bit is altered between transmission and reception
 - Binary 1 is transmitted and binary 0 is received
 - Binary 0 is transmitted and binary 1 is received

Single bit errors

Isolated error that alters one bit but does not affect nearby bits

Can occur in the presence of white noise

Burst errors

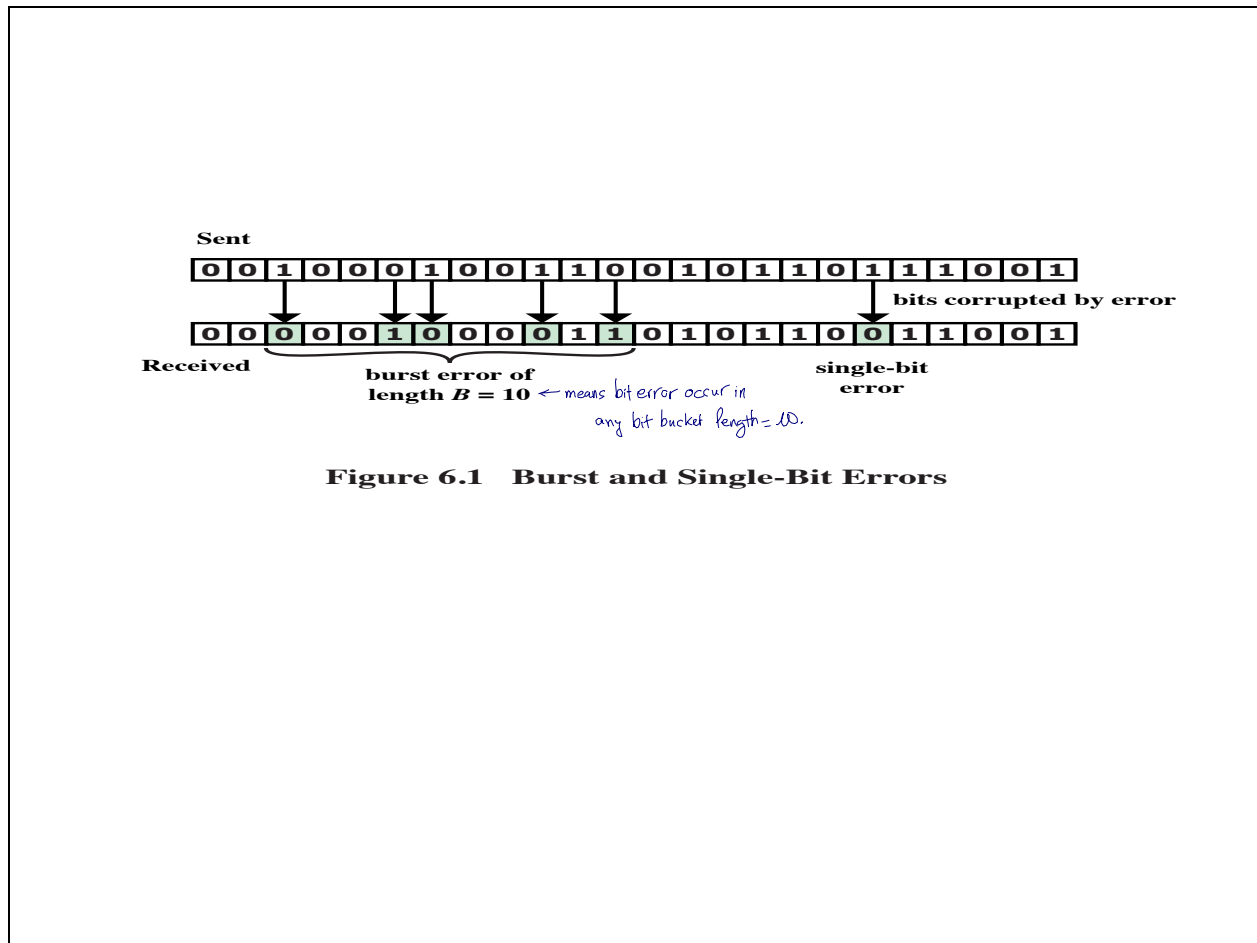
Contiguous sequence of B bits in which the first and last bits and any number of intermediate bits are received in error

Can be caused by impulse noise or by fading in a mobile wireless environment

Effects of burst errors are greater at higher data rates

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.



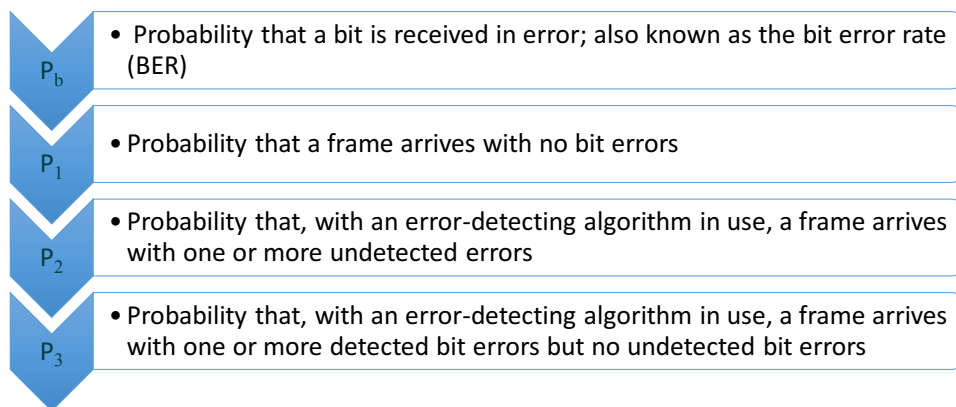
Thus, in an error burst, there is a cluster of bits in which a number of errors occur, although not necessarily all of the bits in the cluster suffer an error. Figure 6.1 provides an example of both types of errors.

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.

Error Detection

- Regardless of design you will have errors, resulting in the change of one or more bits in a transmitted frame
- Frames
 - Data transmitted as one or more contiguous sequences of bits



- The probability that a frame arrives with no bit errors decreases when the probability of a single bit error increases
- 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

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

false negative

True Positive

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.

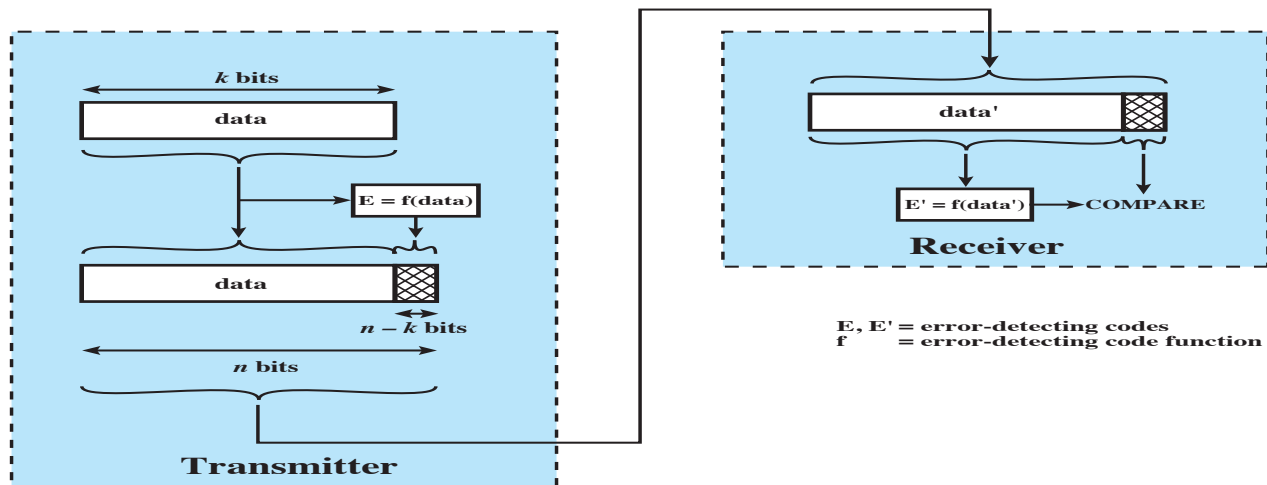


Figure 6.2 Error Detection Process

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.

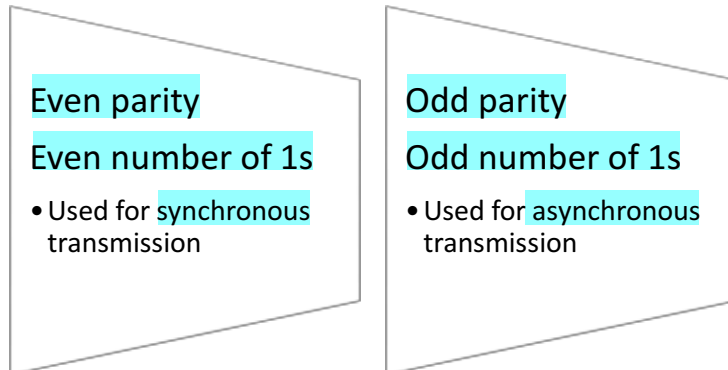
The receiver separates the incoming frame into the k bits of data and $(n - k)$ bits of the error-detecting code. The receiver performs the same error-detecting calculation on the data bits and compares this value with the value of the incoming error-detecting code. A detected error occurs if and only 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.

Parity Check

- The simplest error detecting scheme is to append a parity bit to the end of a block of data



- If any even number of bits are inverted due to error, an undetected error occurs

The simplest error-detecting scheme is to append a parity bit to the end of a block of data. 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). 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.

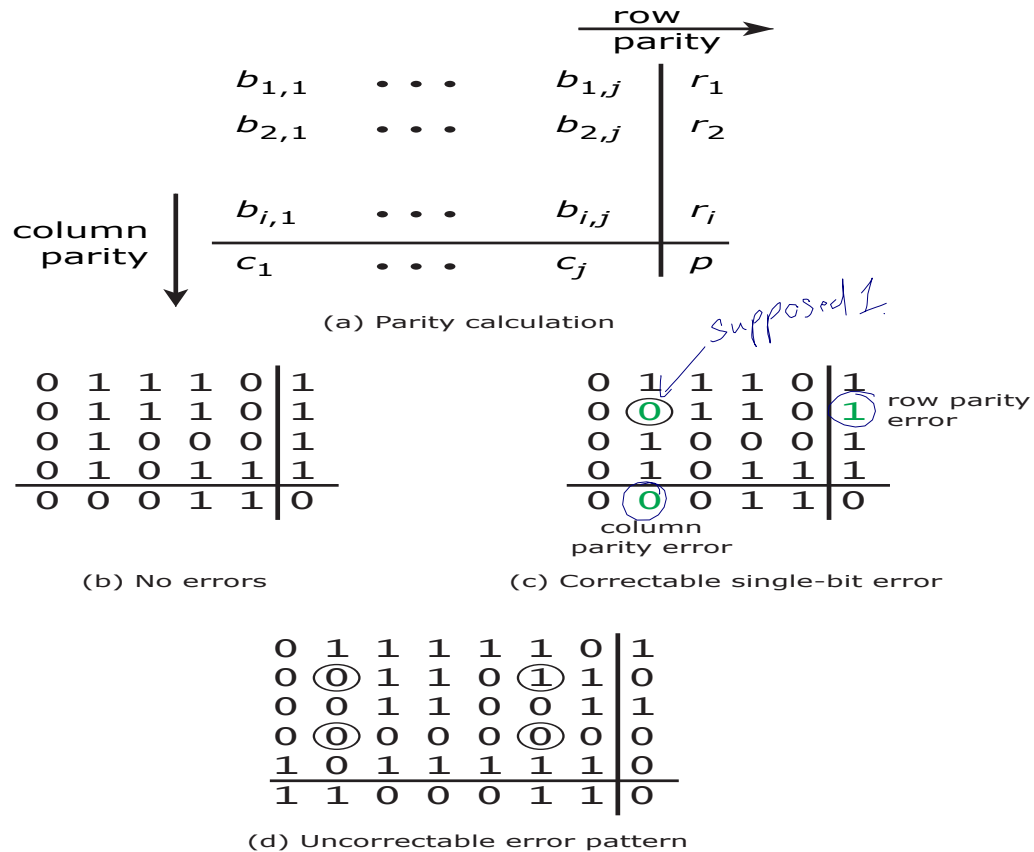


Figure 6.3 A Two-Dimensional Even Parity Scheme


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.

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.

The Internet Checksum

- Error detecting code used in many Internet standard protocols, including IP, TCP, and UDP
- Ones-complement operation 
 - Replace 0 digits with 1 digits and 1 digits with 0 digits
- Ones-complement addition
 - The two numbers are treated as unsigned binary integers and added
 - If there is a carry out of the leftmost bit, add 1 to the sum (end-around carry)

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. 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.

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.

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

(a) Checksum calculation by sender

(b) Checksum verification by receiver

Figure 6.4 Example of Internet Checksum

Receiver.

$$\begin{array}{r}
 00\ 01\ F2\ 03\ F4\ F5\ F6\ F7\ 0D \\
 \hline
 F204 \quad F4F5 \\
 \hline
 E6FA \quad F6F7 \\
 \hline
 DDF2 \\
 \text{ones complement} \rightarrow 220D \\
 \text{result}
 \end{array}$$

$$\begin{array}{r}
 DDF2 \\
 + 220D \\
 \hline
 FFFF \checkmark
 \end{array}$$

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.

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.

not study yet.

Cyclic Redundancy Check (CRC)

- One of the most common and powerful error-detecting codes
- Given a k bit block of bits, the transmitter generates an $(n - k)$ bit frame check sequence (FCS) which is exactly divisible by some predetermined number
- Receiver divides the incoming frame by that number
 - If there is no remainder, assume there is no error

One of the most common, and one of the most powerful, error-detecting codes is the cyclic redundancy check (CRC), 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.

Can state this procedure in three equivalent ways: modulo 2 arithmetic, polynomials, and digital logic.

CRC Process

- Modulo 2 arithmetic
 - Uses binary addition with no carries
 - An example is shown on page 194 in the textbook
- Polynomials
 - Express all values as polynomials in a dummy variable X , with binary coefficients
 - Coefficients correspond to the bits in the binary number
 - An example is shown on page 197 in the textbook
- Digital logic
 - Dividing circuit consisting of XOR gates and a shift register
 - 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
 - An example is referenced on page 199 in the textbook

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.

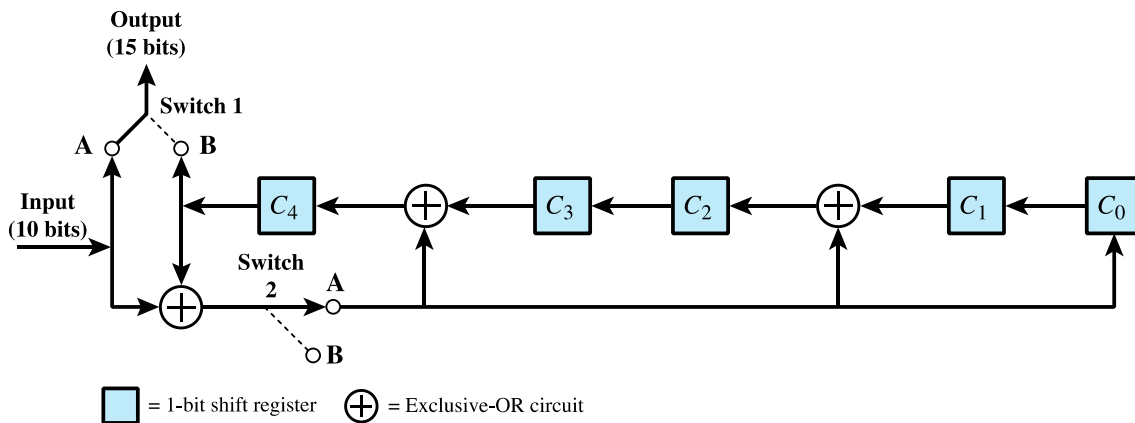
A second way of viewing the CRC process is to express all values as polynomials in a dummy variable X , with binary coefficients. The coefficients correspond to the bits in the binary number.

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.

$$\begin{array}{r}
 P(X) \rightarrow X^5 + X^4 + X^2 + 1 \overline{) \begin{array}{l} X^9 + X^8 + X^6 + X^4 + X^2 + X \\ X^{14} \quad X^{12} \quad X^8 + X^7 + X^5 \end{array} } \leftarrow Q(X) \\
 \underline{X^{14} + X^{13} + X^{11} + X^9} \quad \leftarrow X^5 D(X) \\
 X^{13} + X^{12} + X^{11} + X^9 + X^8 \\
 \underline{X^{13} + X^{12} + X^{10} + X^8} \\
 X^{11} + X^{10} + X^9 + X^7 \\
 \underline{X^{11} + X^{10} + X^8 + X^6} \\
 X^9 + X^8 + X^7 + X^6 + X^5 \\
 \underline{X^9 + X^8 + X^6 + X^4} \\
 X^7 + X^5 + X^4 \\
 \underline{X^7 + X^6 + X^4 + X^2} \\
 X^6 + X^5 + X^2 \\
 \underline{X^6 + X^5 + X^3 + X} \\
 X^3 + X^2 + X \leftarrow R(X)
 \end{array}$$

Figure 6.5 Example of Polynomial Division

Figure 6.5 shows the polynomial division that corresponds to the binary division in the preceding example.



(a) Shift-register implementation

	C_4	C_3	C_2	C_1	C_0	$C_4 \approx C_3 \approx I$	$C_4 \approx C_1 \approx I$	$C_4 \approx I$	$I = \text{input}$	
Initial	0	0	0	0	0	1	1	1	1	} Message to be sent
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	
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 $X^5 + X^4 + X^2 + 1$

The architecture of a CRC circuit is best explained by first considering an example, which is illustrated in Figure 6.6.

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.

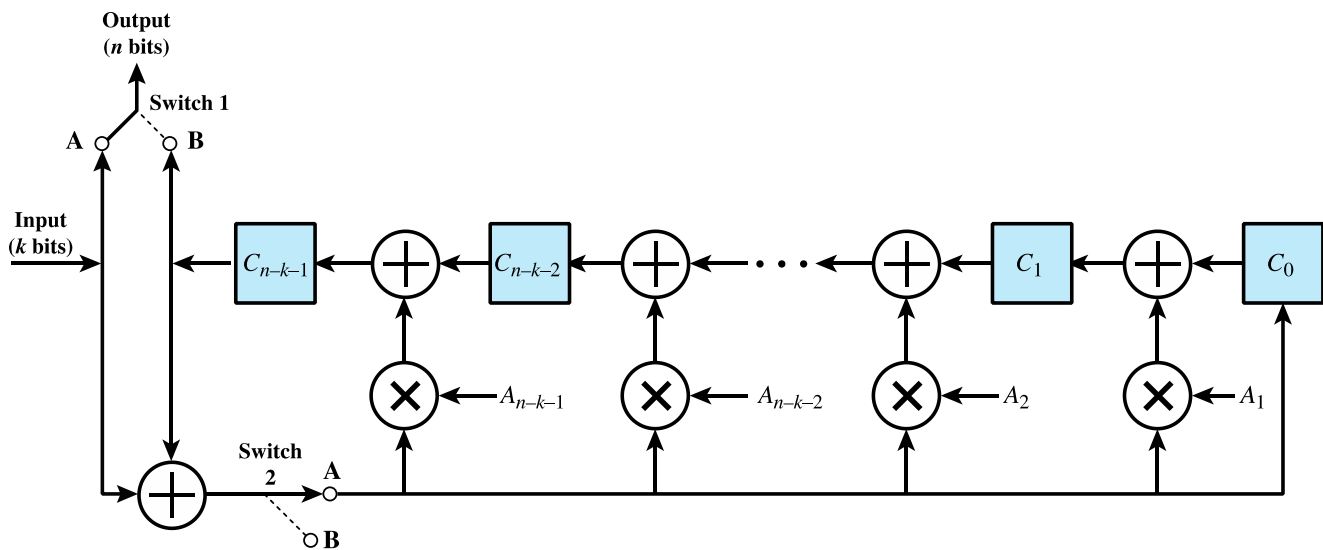


Figure 6.7 General CRC Architecture to Implement Divisor
 $(1 + A_1X + A_2X^2 + \dots + A_{n-k-1}X^{n-k-1} + X^{n-k})$

Figure 6.7 indicates the general architecture of the shift-register implementation of a CRC for a polynomial.

Forward Error Correction

- Correction of detected errors usually requires data blocks to be retransmitted
- Not appropriate for wireless applications:
 - The bit error rate (BER) on a wireless link can be quite high, which would result in a large number of retransmissions
 - Propagation delay is very long compared to the transmission time of a single frame
- Need to correct errors on basis of bits received

Codeword
<ul style="list-style-type: none"> • On the transmission end each k-bit block of data is mapped into an n-bit block ($n > k$) using a forward error correction (FEC) encoder

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.

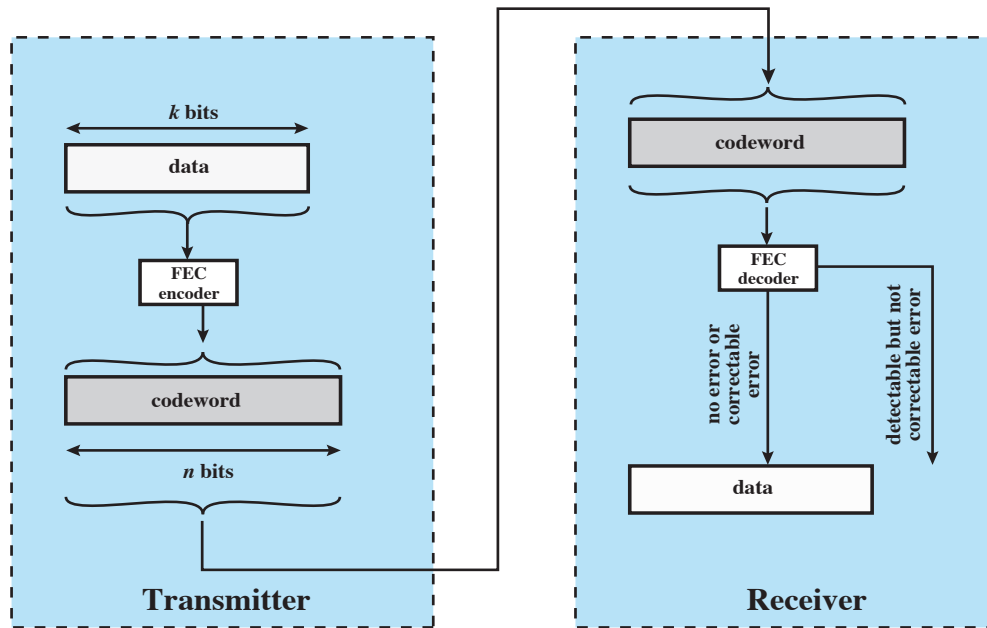


Figure 6.8 Error Correction Process

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.
- **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.

Block Code Principles

- 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
 - See example on page 203 in the textbook
- Redundancy of the code
 - The ratio of redundant bits to data bits $(n-k)/k$
- Code rate
 - The ratio of data bits to total bits k/n
 - Is a measure of how much additional bandwidth is required to carry data at the same data rate as without the code
 - See example on page 205 in the textbook

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.

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.

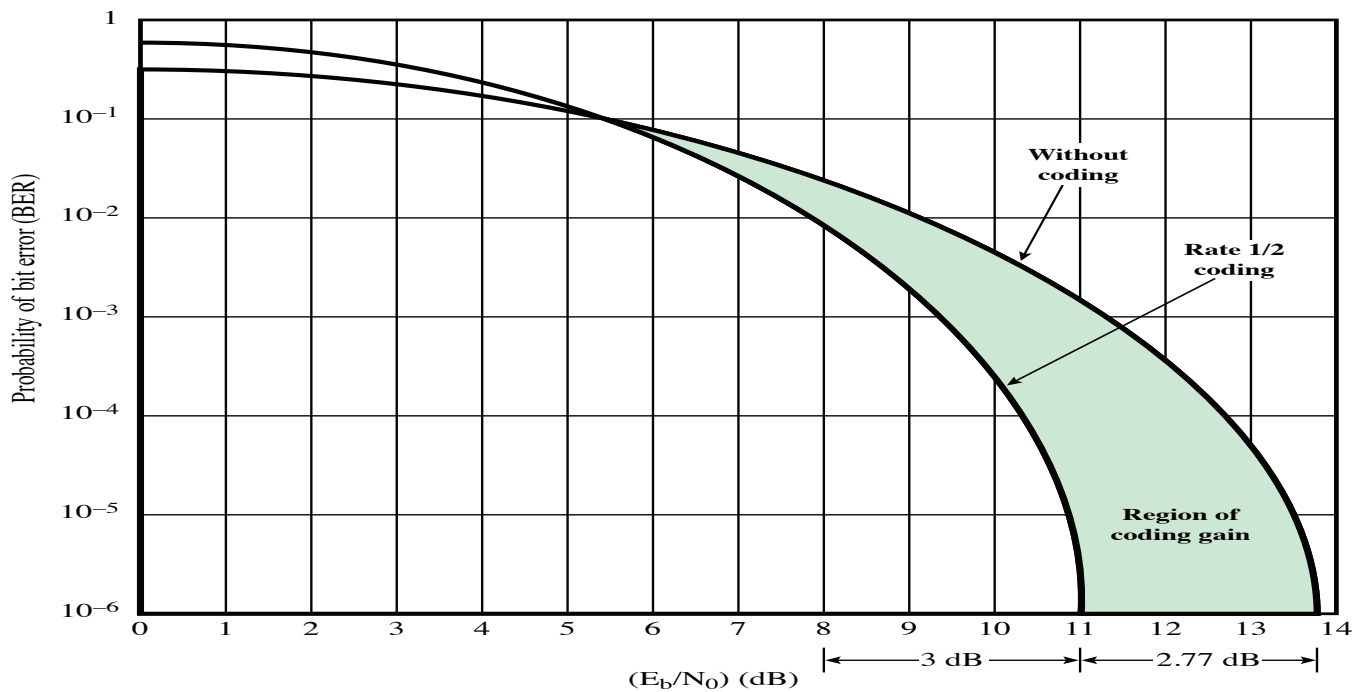


Figure 6.9 How Coding Improves System Performance

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 encoding schemes.

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.

Summary

- Types of errors
- Error detection
- Parity check
 - Parity bit
 - Two-dimensional parity check
- Internet checksum
- Cyclic redundancy check
 - Modulo 2 arithmetic
 - Polynomials
 - Digital logic
- Forward error correction
 - Block code principles

Chapter 6 summary.