Chapter 10

Error Detection and Correction

Introduction

- Networks must be able to transfer data from one device to another with acceptable accuracy.
- Any time data are transmitted from one node to the next, they can become corrupted in passage.
- At the data-link layer, if a frame is corrupted between the two nodes, it needs to be corrected before it continues its journey to other nodes.
- However, most link-layer protocols simply discard the frame and let the upper-layer protocols handle the retransmission of the frame.
- Some multimedia applications, however, try to correct the corrupted frame.

Note

Data can be corrupted
during transmission.

Some applications require that
errors be detected and corrected.

10-1 INTRODUCTION

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

Topics discussed in this section:

- Types of Errors
  - Redundancy
  - Détection Versus Correction
  - Forward Error Correction Versus Retransmission
  - Coding

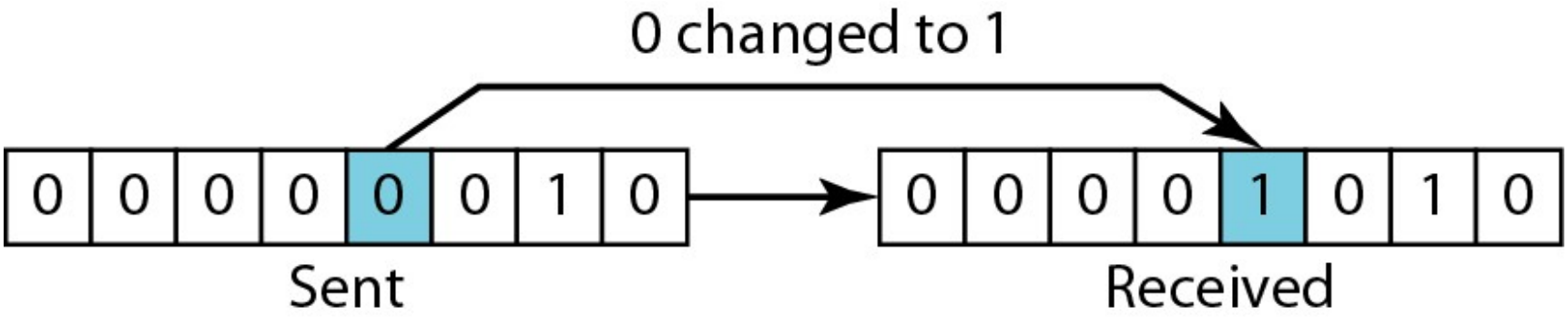
10-1 INTRODUCTION

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

Note

In a single-bit error, only 1 bit in the data unit has changed.

Figure 10.1 *Single-bit error*



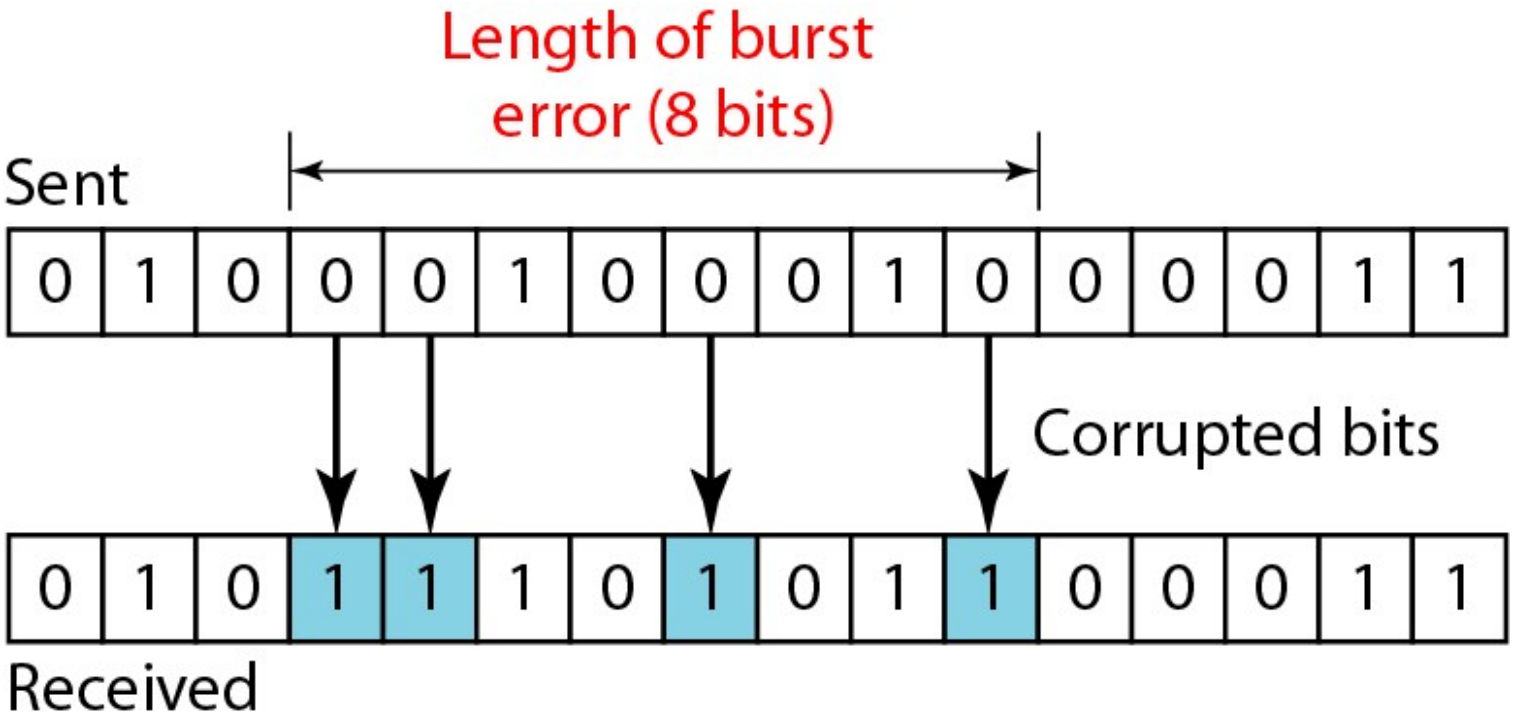
Note

A burst error means that 2 or more bits in the data unit have changed.

Types of Errors

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

Figure 10.2 *Burst error of length 8*



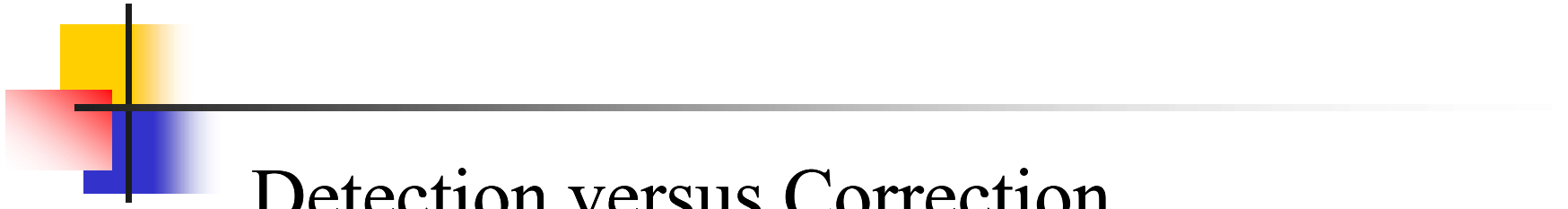
- Redundancy
- The central concept in detecting or correcting errors is redundancy. To be able to detect or correct errors, we need to send some extra bits with our data.
- These redundant bits are added by the sender and removed by the receiver. Their presence allows the receiver to detect or correct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

Note

To detect or correct errors, we need to send extra (redundant) bits with data.

Detection versus Correction

- In error detection, we are only looking to see if any error has occurred.
- The answer is a simple yes or no. We are not even interested in the number of corrupted bits.
- A single-bit error is the same for us as a burst error. In error correction, we need to know the exact number of bits that are corrupted and, more importantly, their location in the message.
- The number of errors and the size of the message are important factors



Detection versus Correction

- If we need to correct a single error in an 8-bit data unit, we need to consider eight possible error locations; if we need to correct two if we need to correct two errors in a data unit of the same size, we need to consider 28 possibilities.
- Imagine the receiver's difficulty in finding 10 errors in a data unit of 1000 bits.

Detection versus Correction

- Forward Error Correction Versus Retransmission
- Forward error correction is the process in which the receiver tries to guess the message by using redundant bits (if the number of errors is small).
- Correction by retransmission is a technique in which the receiver detects the occurrence of an error and asks the sender to resend the message.
- Resending is repeated until a message arrives that the receiver believes is error-free

Detection versus Correction

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

BLOCK CODING

- In block coding, we divide our message into blocks, each of k bits, called datawords.
- We add r redundant bits to each block to make the length n = k + r. The resulting n-bit blocks are called codewords.
- We have a set of datawords, each of size k, and a set of codewords, each of size of n.

BLOCK CODING

- With k bits, we can create a combination of 2^k datawords; with n bits, we can create a combination of 2^n codewords.
- Since n > k, the number of possible codewords is larger than the number of possible datawords.
- The block coding process is one-to-one; the same dataword is always encoded as the same codeword.
- This means that we have $2^n - 2^k$ codewords that are not used. We call these codewords invalid or illegal.
- The trick in error detection is the existence of these invalid codes.

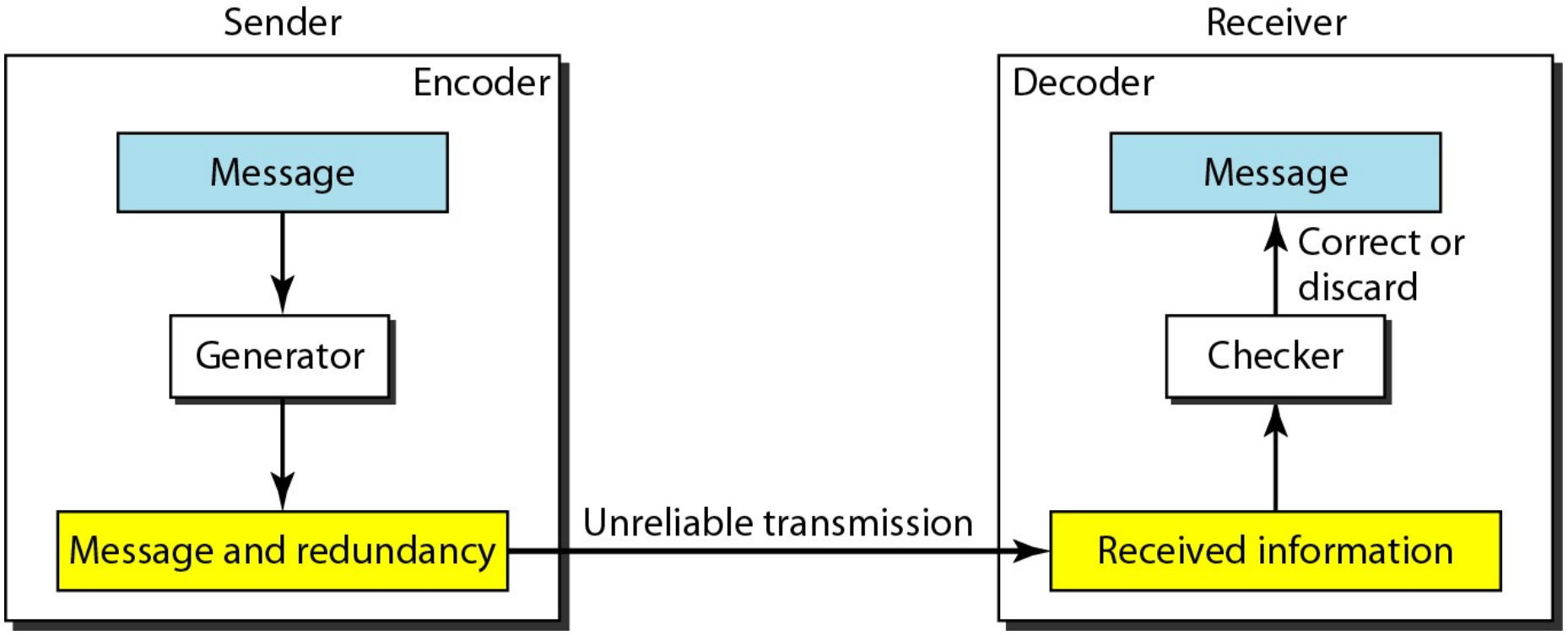
BLOCK CODING

- Error Detection
- How can errors be detected by using block coding? If the following two conditions are met, the receiver can detect a change in the original codeword.
- 1. The receiver has (or can find) a list of valid codewords.
- 2. The original codeword has changed to an invalid one.

BLOCK CODING

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

Figure 10.3 *The structure of encoder and decoder*



Example 10.1

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

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

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

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

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

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

10-2 BLOCK CODING

In block coding, we divide our message into blocks, each of k bits, called datawords. We add r redundant bits to each block to make the length n = k + r. The resulting n-bit blocks are called codewords.

Topics discussed in this section:

Error Detection
Error Correction
Hamming Distance
Minimum Hamming Distance

Figure 10.5 *Datawords and codewords in block coding*

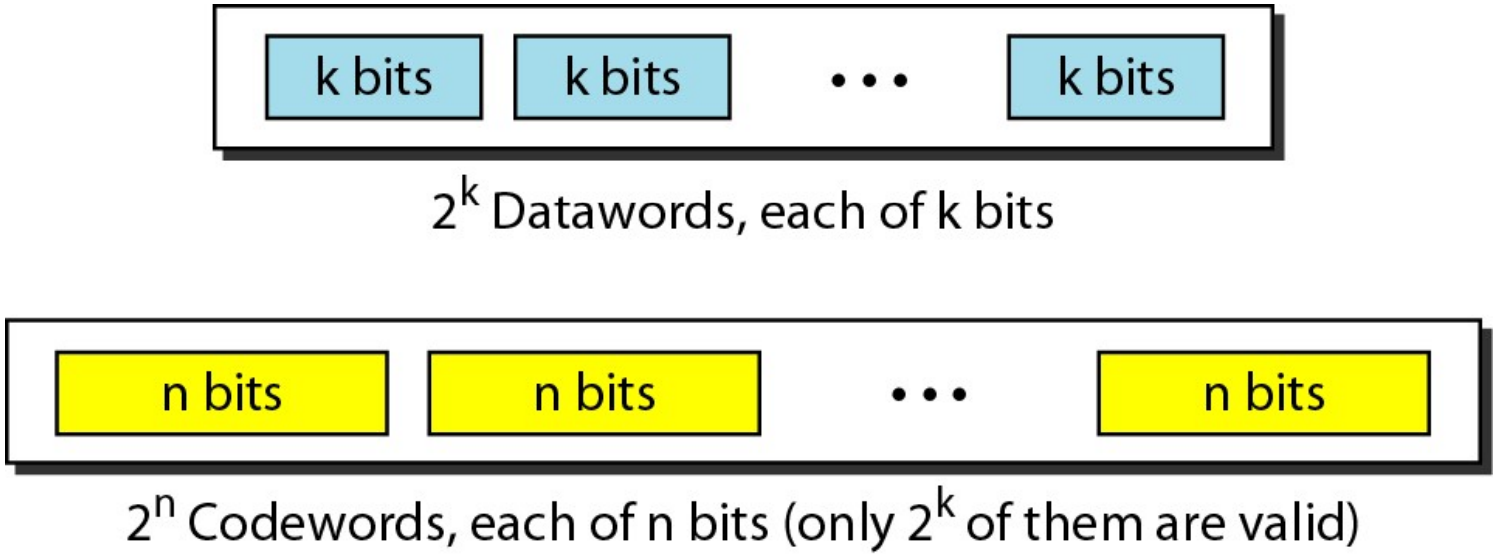
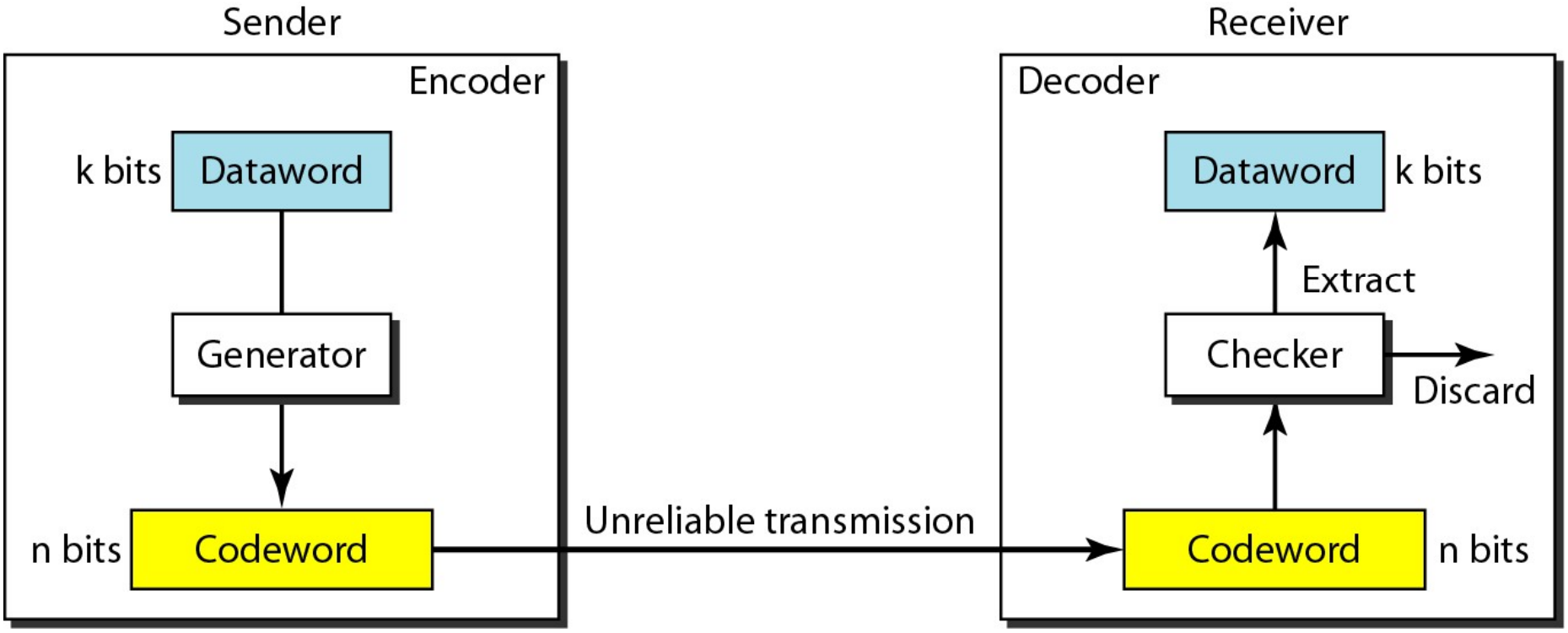
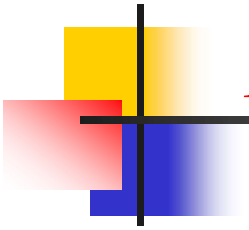


Figure 10.6 *Process of error detection in block coding*



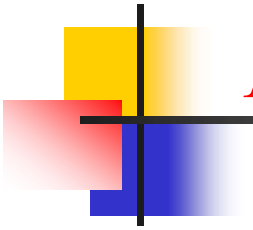


Example 10.2

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

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

- *1. The receiver receives 011. It is a valid codeword. The receiver extracts the dataword 01 from it.*



Example 10.2 (continued)

2. *The codeword is corrupted during transmission, and 111 is received. This is not a valid codeword and is discarded.*

3. *The codeword is corrupted during transmission, and 000 is received. This is a valid codeword. The receiver incorrectly extracts the dataword 00. Two corrupted bits have made the error undetectable.*

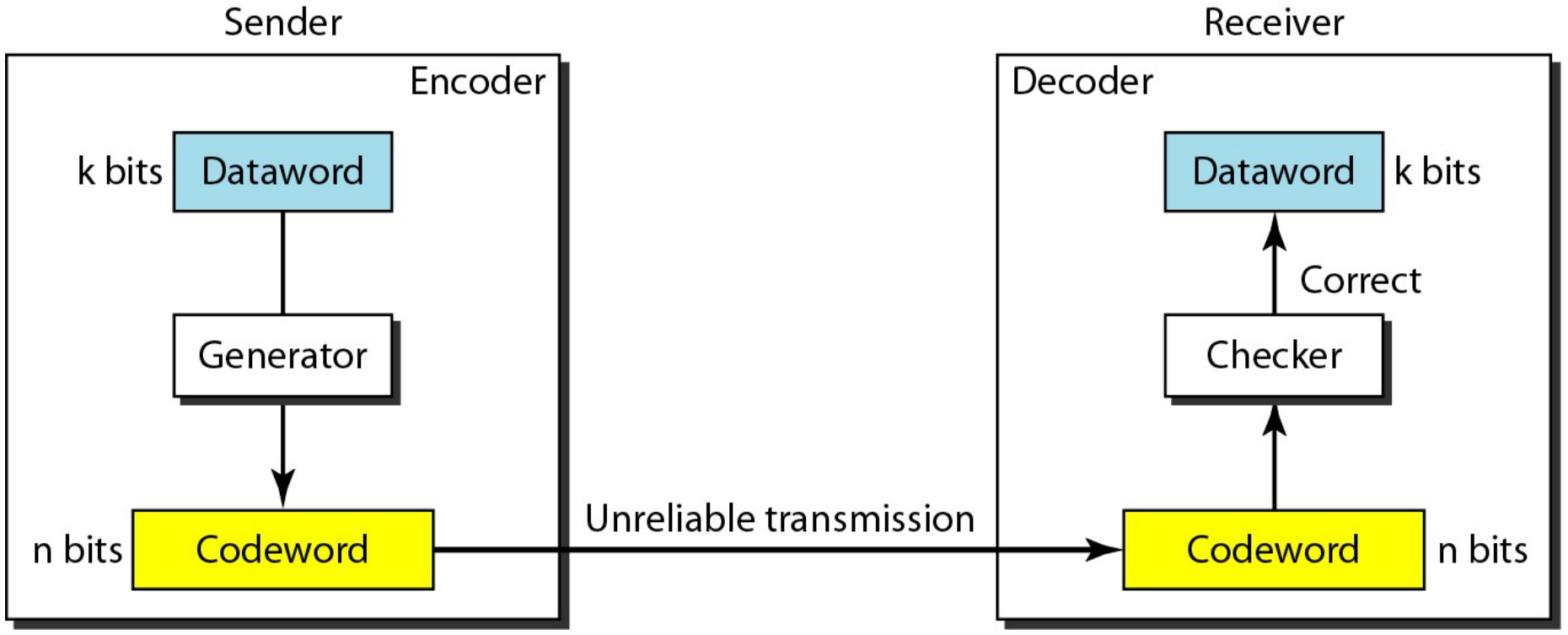
Table 10.1 *A code for error detection (Example 10.2)*

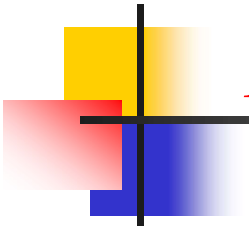
Datawords	*Codewords*
00	000
01	011
10	101
11	110

Note

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

Figure 10.7 *Structure of encoder and decoder in error correction*





Example 10.3

- ***Let us add more redundant bits to Example 10.2 to see if the receiver can correct an error without knowing what was actually sent.***
- ***We add 3 redundant bits to the 2-bit dataword to make 5-bit codewords. Table 10.2 shows the datawords and codewords. Assume the dataword is 01.***
- ***The sender creates the codeword 01011. The codeword is corrupted during transmission, and 01001 is received.***
- ***First, the receiver finds that the received codeword is not in the table. This means an error has occurred.***
- ***The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.***

Example 10.3 (continued)

1. ***Comparing the received codeword with the first codeword in the table (01001 versus 00000), the receiver decides that the first codeword is not the one that was sent because there are two different bits.***

2. ***By the same reasoning, the original codeword cannot be the third or fourth one in the table.***

3. ***The original codeword must be the second one in the table because this is the only one that differs from the received codeword by 1 bit. The receiver replaces 01001 with 01011 and consults the table to find the dataword 01.***

Table 10.2 *A code for error correction (Example 10.3)*

Dataword	*Codeword*
00	00000
01	01011
10	10101
11	11110

- Hamming Distance
- The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits.
- We show the Hamming distance between two words x and y as d(x, y)
- For example, if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Ham ing distance between the two is d(00000, 01101) = 3. In other words, if the Hamming distance between the sent and the received codeword is not zero, the codeword has been corrupted during transmission.

Note

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

Example 10.4

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

1. The Hamming distance d(000, 011) is 2 because

$$000 \oplus 011 \text{ is } 011 \text{ (two 1s)}$$

2. The Hamming distance d(10101, 11110) is 3 because

$$10101 \oplus 11110 \text{ is } 01011 \text{ (three 1s)}$$

- **Minimum Hamming Distance for Error Detection**
- **In a set of codewords, the minimum Hamming distance is the smallest Hamming distance between all possible pairs of codewords.**
- **If s errors occur during transmission, the Hamming distance between the sent codeword and received codeword is s. If our system is to detect up to s errors, the minimum distance between the valid codes must be (s + 1), so that the received codeword does not match a valid codeword.**
- **Although a code with dmin = s + 1 may be able to detect more than s errors in some special cases, only s or fewer errors are guaranteed to be detected.**

Note

The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.

Example 10.5

Find the minimum Hamming distance of the coding scheme in Table 10.1.

Solution

We first find all Hamming distances.

$d(000, 011) = 2$	$d(000, 101) = 2$	$d(000, 110) = 2$	$d(011, 101) = 2$
$d(011, 110) = 2$	$d(101, 110) = 2$		

The d_{min} in this case is 2.

Example 10.6

Find the minimum Hamming distance of the coding scheme in Table 10.2.

Solution

We first find all the Hamming distances.

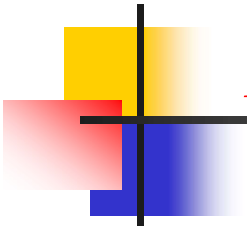
$d(00000, 01011) = 3$	$d(00000, 10101) = 3$	$d(00000, 11110) = 4$
$d(01011, 10101) = 4$	$d(01011, 11110) = 3$	$d(10101, 11110) = 3$

The d_{min} in this case is 3.

Note

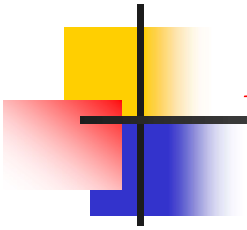
To guarantee the detection of up to s errors

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



Example 10.7

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



Example 10.8

- ***Our second block code scheme (Table 10.2) has $d_{min} = 3$. This code can detect up to two errors.***
- ***Again, we see that when any of the valid codewords is sent, two errors create a codeword which is not in the table of valid codewords. The receiver cannot be fooled.***

- ***However, some combinations of three errors change a valid codeword to another valid codeword. The receiver accepts the received codeword and the errors are undetected.***

- **Minimum Hamming Distance for Error Correction**
- **When a received codeword is not a valid codeword, the receiver needs to decide which valid codeword was actually sent.**
- **The decision is based on the concept of territory, an exclusive area surrounding the codeword. Each valid codeword has its own territory.**

Note

To guarantee correction of up to *t* errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = 2t + 1$.

Example 10.9

- ***A code scheme has a Hamming distance $d_{min} = 4$. What is the error detection and correction capability of this scheme?***

Solution

This code guarantees the detection of up to three errors ($s = 3$), but it can correct up to one error.

In other words, if this code is used for error correction, part of its capability is wasted.

Error correction codes need to have an odd minimum distance (3, 5, 7, . . .).

10-3 LINEAR BLOCK CODES

- ***Almost all block codes used today belong to a subset called linear block codes.***
- ***The use of nonlinear block codes for error detection and correction is difficult because their structure makes theoretical analysis and implementation difficult.***
- ***A linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.***

Topics discussed in this section:

Minimum Distance for Linear Block Codes
Some Linear Block Codes

Example 10.10

- *Let us see if the two codes we defined in Table 10.1 and Table 10.2 belong to the class of linear block codes.*

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

2. *The scheme in Table 10.2 is also a linear block code. We can create all four codewords by XORing two other codewords.*

Note

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

- ***Minimum Distance for Linear Block Codes***
- **It is simple to find the minimum Hamming distance for a linear block code. The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s.**
- **Example 10.11**
- **In our first code (Table 10.1), the numbers of Is in the nonzero codewords are 2, 2, and 2. So the minimum Hamming distance is $d_{min} = 2$.**
- **In our second code (Table 10.2), the numbers of Is in the nonzero codewords are 3, 3, and 4. So in this code we have $d_{min} = 3$.**

Example 10.10

- Some Linear Block Codes
- **Let us now show some linear block codes.**
- **Parity-Check Code**
- **The most familiar error-detecting code is the parity-check code. This code is a linear block code.**
- **In this code, a k-bit dataword is changed to an n-bit codeword where n = k + 1.**
- **The extra bit, called the parity bit, is selected to make the total number of 1s in the codeword even.**
- **Our first code (Table 10.1) is a parity-check code (k = 2 and n = 3). The code in Table 10.2 is also a parity-check code with k = 4 and n = 5.**

Note

A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{min} = 2$.

Figure 10.10 shows a possible structure of an encoder (at the sender) and a decoder (at the receiver).

Table 10.3 *Simple parity-check code C(5, 4)*

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

Example 10.10

- **Parity-Check Code**
- **Figure 10.10 shows a possible structure of an encoder (at the sender) and a decoder (at the receiver).**
- **The encoder uses a generator that takes a copy of a 4-bit dataword and generates a parity bit r_{oo}**
- **The dataword bits and the parity bit create the 5-bit codeword. The parity bit that is added makes the number of Is in the codeword even.**
- **This is normally done by adding the 4 bits of the dataword (modulo-2); the result is the parity bit. In other words,**

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

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

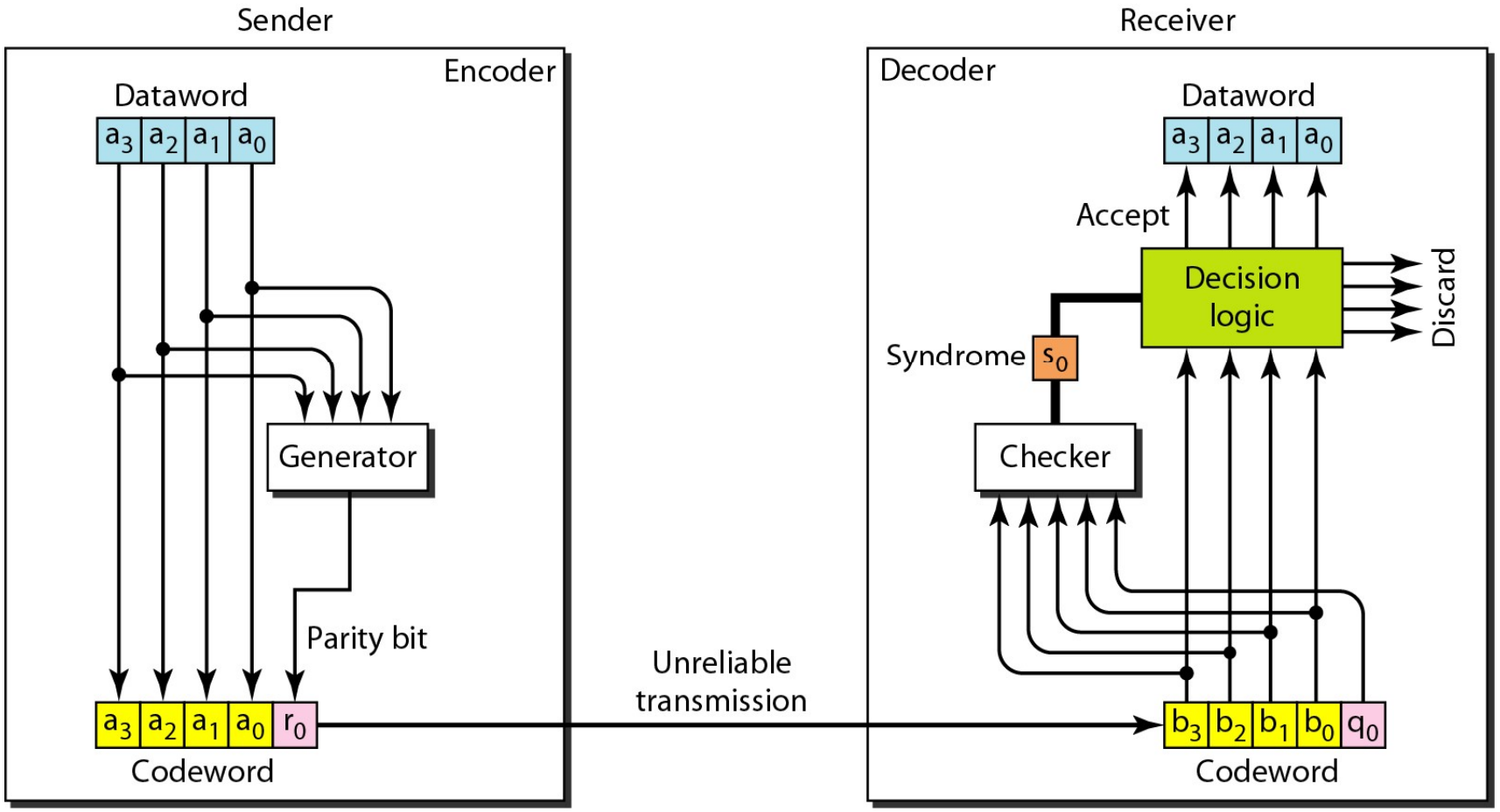
Example 10.10

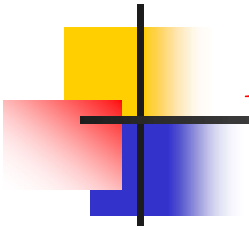
- **Parity-Check Code**
- **The sender sends the codeword which may be corrupted during transmission.**
- **The receiver receives a 5-bit word. The checker at the receiver does the same thing : The addition is done over all 5 bits.**
- **The result, which is called the syndrome, is just 1 bit. The syndrome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

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

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

Figure 10.10 *Encoder and decoder for simple parity-check code*





Example 10.12

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

1. *No error occurs; the received codeword is 10111. The syndrome is 0. The dataword 1011 is created.*
2. *One single-bit error changes a_1. The received codeword is 10011. The syndrome is 1. No dataword is created.*
3. *One single-bit error changes r_0 . The received codeword is 10110. The syndrome is 1. No dataword is created.*

Example 10.12 (continued)

4. *An error changes r_0 and a second error changes a_3. The received codeword is 00110. The syndrome is 0.*
   The dataword 0011 is created at the receiver. Note that here the dataword is wrongly created due to the syndrome value.
5. *Three bits—a_3, a_2, and a_1—are changed by errors. The received codeword is 01011. The syndrome is 1. The dataword is not created.*

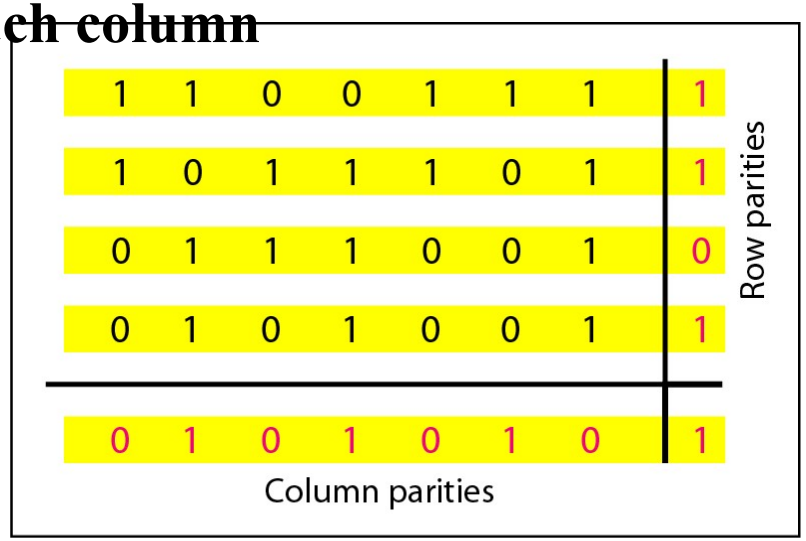
This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

Note

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

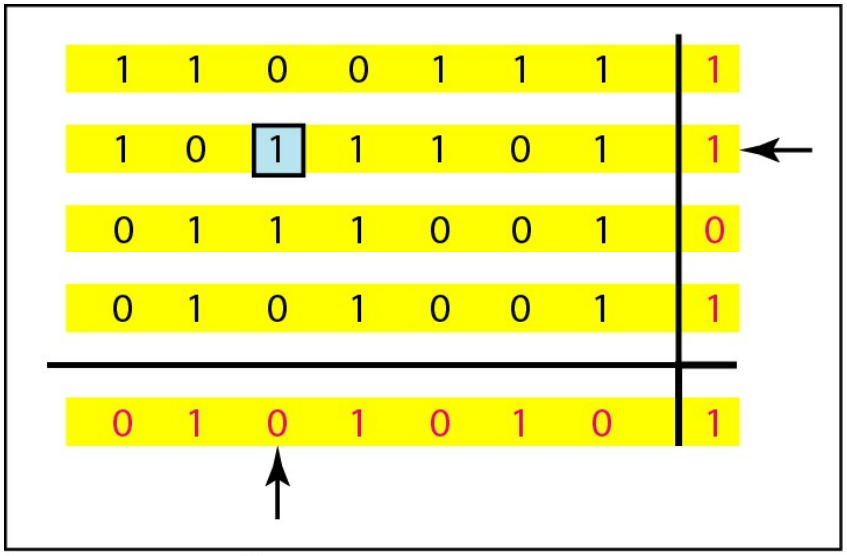
Figure 10.11 *Two-dimensional parity-check code*

- **A better approach is the two-dimensional parity check. In this method, the dataword is organized in a table (rows and columns).**
- **Five 7-bit bytes, are put in separate rows. For each row and each column, 1 parity-check bit is calculated.**
- **The whole table is then sent to the receiver, which finds the syndrome for each row and each column**

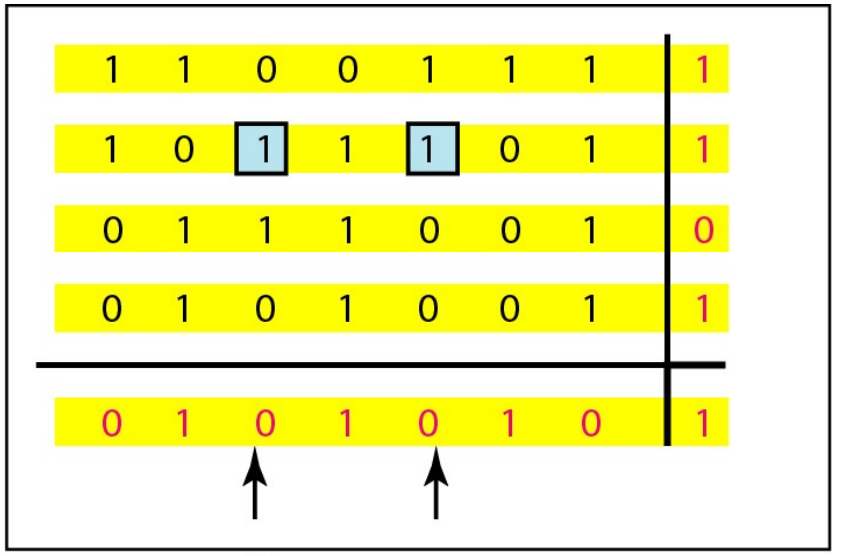


a. Design of row and column parities

Figure 10.11 *Two-dimensional parity-check code*

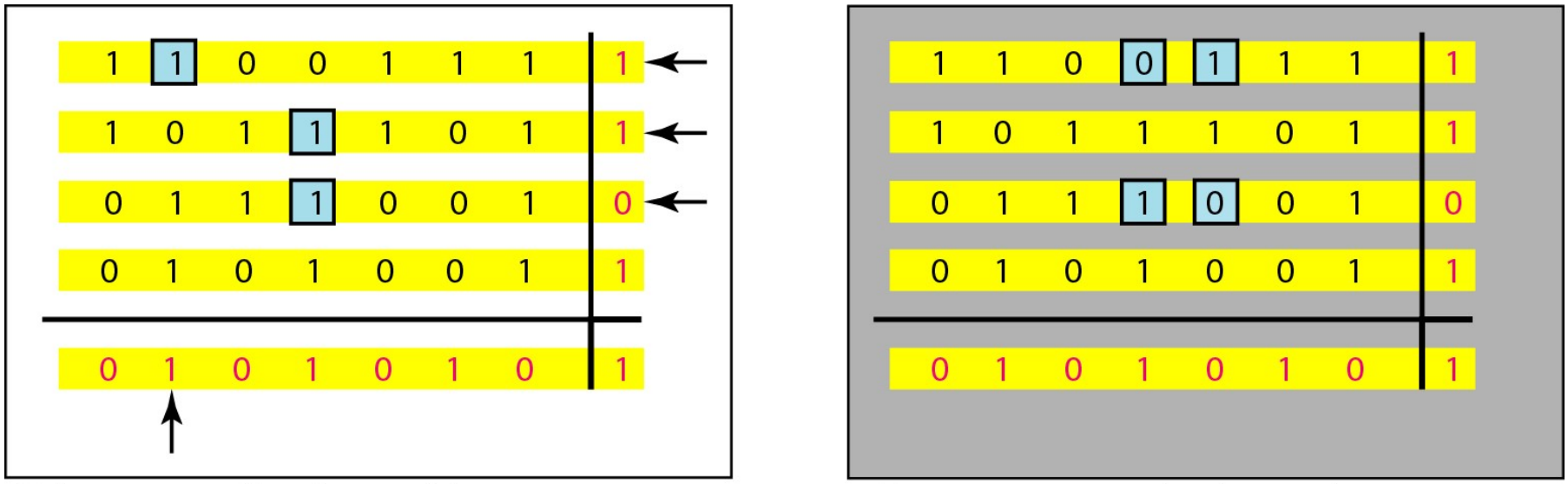


b. One error affects two parities



c. Two errors affect two parities

Figure 10.11 *Two-dimensional parity-check code*



d. Three errors affect four parities

e. Four errors cannot be detected

- **The two-dimensional parity check can detect up to three errors that occur anywhere in the table.**

- **Hamming Codes**
- **Hamming codes were originally designed with $d_{min} = 3$, which means that they can detect up to two errors or correct one single error. Although there are some Hamming codes that can correct more than one error,**

First let us find the relationship between *n* and *k* in a Hamming code. We need to choose an integer $m >= 3$. The values of nand *k* are then calculated from mas $n = 2^m - 1$ and $k ::: n - m$. The number of check bits $r = m$.

For example, if $m = 3$, then $n ::: 7$ and $k::: 4$. This is a Hamming code $C(7, 4)$ with $d_{min} = 3$. Table 10.4 shows the datawords and codewords for this code.

Note

All Hamming codes discussed in this book have $d_{min} = 3$.

The relationship between *m* and *n* in these codes is $n = 2^m - 1$.

- **Hamming Codes**
- **A copy of a 4-bit dataword is fed into the generator that creates three parity checks r_o, r_1' and r_2' as shown below:**

$$T_0 = a_2 + a_1 + a_o \qquad \text{modulo-2}$$
$$T_1 = a_3 + a_2 + a_1 \qquad \text{modulo-2}$$
$$T_2 = a_1 + a_0 + a_3 \qquad \text{modulo-2}$$

- **These three equations are not the unique ones; any three equations that involve 3 of the 4 bits in the dataword and create independent equations (a combination of two cannot create the third) are valid.**

- **Hamming Codes**
- **The checker in the decoder creates a 3-bit syndrome (s2s1s0) in which each bit is the parity check for 4 out of the 7 bits in the received codeword:**

$$S_0 = b_2 + b_1 + b_o + q_o \qquad \text{modulo-2}$$
$$S_l = b_3 + b_2 + b_1 + q_l \qquad \text{modulo-2}$$
$$S_2 == b_l + b_0 + b_3 + q_2 \qquad \text{modulo-2}$$

Table 10.4 *Hamming code C(7, 4)*

Datawords	*Codewords*	*Datawords*	*Codewords*
0000	0000000	1000	1000110
0001	0001101	1001	1001011
0010	0010111	1010	1010001
0011	0011010	1011	1011100
0100	0100011	1100	1100101
0101	0101110	1101	1101000
0110	0110100	1110	1110010
0111	0111001	1111	1111111

Figure 10.12 *The structure of the encoder and decoder for a Hamming code*

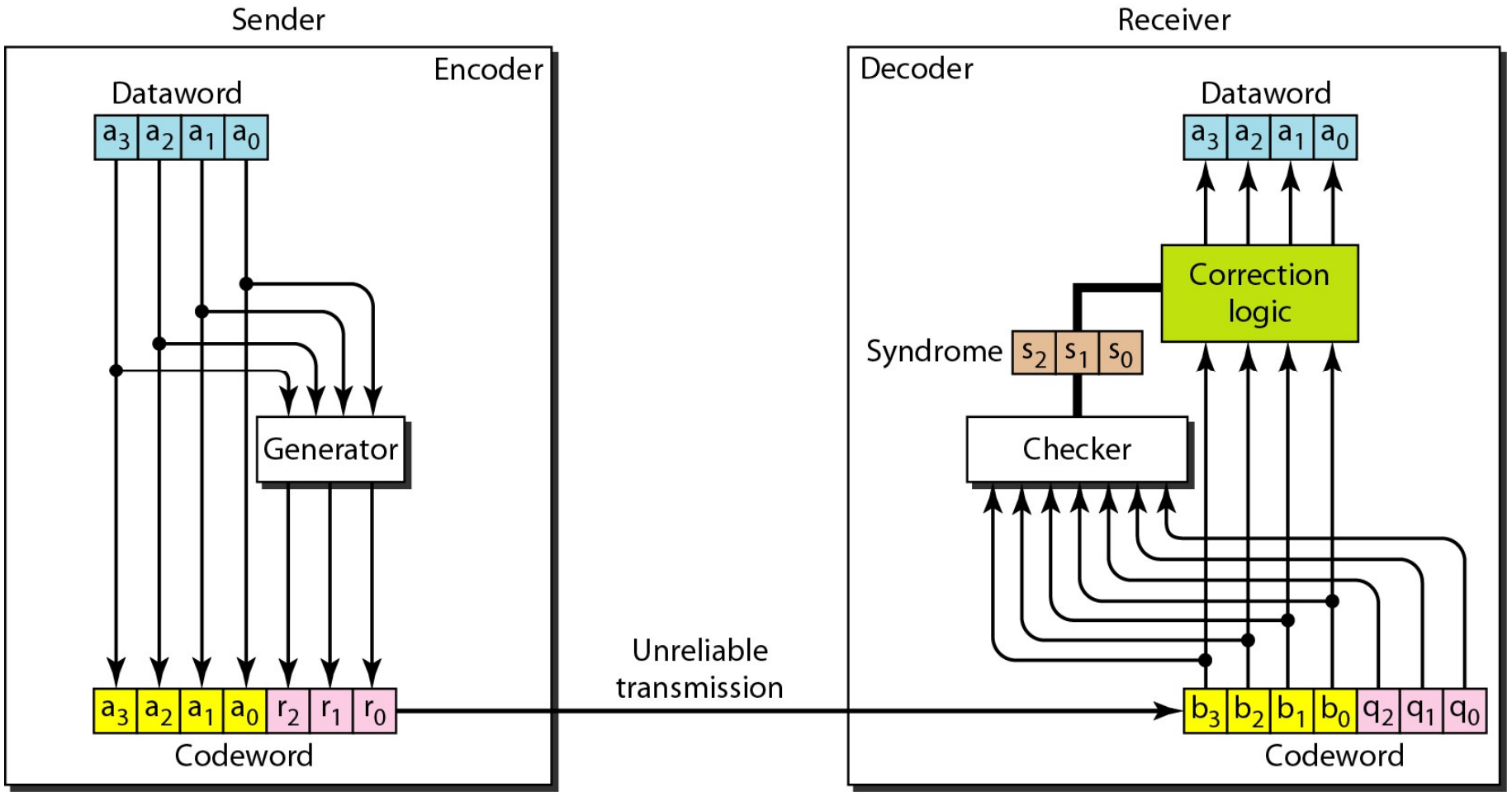
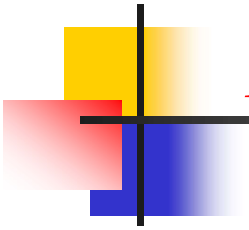


Table 10.5 *Logical decision made by the correction logic analyzer*

Syndrome	000	001	010	011	100	101	110	111
Error	None	q_0	q_1	b_2	q_2	b_0	b_3	b_1



Example 10.13

Let us trace the path of three datawords from the sender to the destination:

1. ***The dataword 0100 becomes the codeword 0100011.***
   The codeword 0100011 is received. The syndrome is 000, the final dataword is 0100.
2. ***The dataword 0111 becomes the codeword 0111001.***
   The syndrome is 011. After flipping b_2 (changing the 1 to 0), the final dataword is 0111.
3. ***The dataword 1101 becomes the codeword 1101000.***
   The syndrome is 101. After flipping b_0, we get 0000, the wrong dataword. This shows that our code cannot correct two errors.

Example 10.14

We need a dataword of at least 7 bits. Calculate values of k and n that satisfy this requirement.

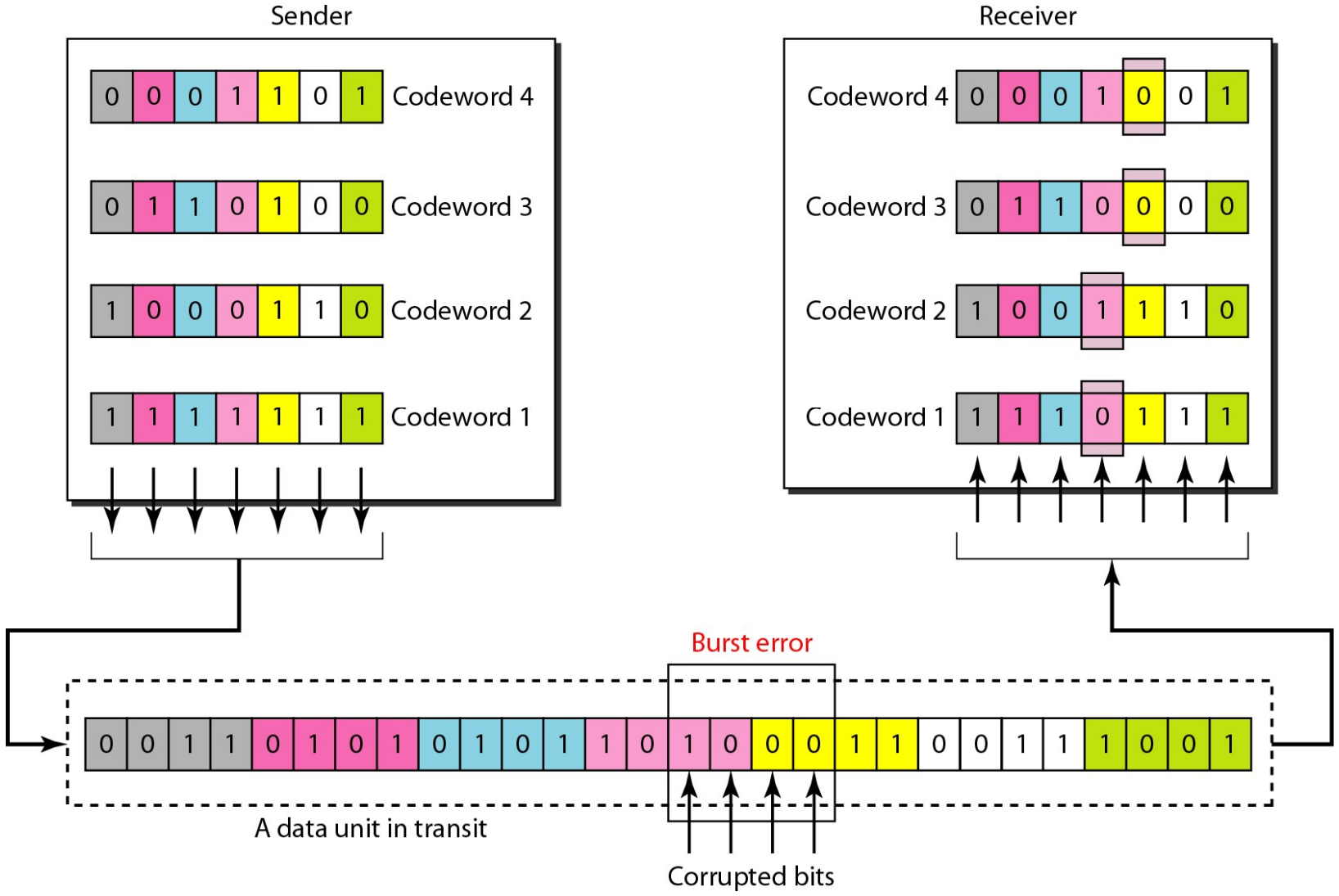
Solution

We need to make $k = n - m$ greater than or equal to 7, or $2^m - 1 - m \geq 7$.

1. ***If we set $m = 3$, the result is $n = 23 - 1$ and $k = 7 - 3$, or 4, which is not acceptable.***
2. ***If we set $m = 4$, then $n = 24 - 1 = 15$ and $k = 15 - 4 = 11$, which satisfies the condition. So the code is***

$C(15, 11)$

Figure 10.13 *Burst error correction using Hamming code*



10-4 CYCLIC CODES

Cyclic codes are special linear block codes with one extra property. In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword.

Topics discussed in this section:

Cyclic Redundancy Check
Hardware Implementation
Polynomials
Cyclic Code Analysis
Advantages of Cyclic Codes
Other Cyclic Codes

10-4 CYCLIC CODES

- **For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword.**
- **In this case, if we call the bits in the first word a0 to a6, and the bits in the second word b0 to b6, we can shift the bits by using the following:**
- **b1 = a0 b2 = a1 b3 = a2 b4 = a3**
- **b5 = a4 b6 = a5 b0 = a6**
- **A subset of cyclic codes called the cyclic redundancy check (CRC), which is used in networks such as LANs and WANs**

Cyclic Redundancy Check

- **In the encoder, the dataword has k bits (4 here); the codeword has n bits (7 here). The size of the dataword is augmented by adding n − k (3 here) 0s to the right-hand side of the word.**
- **The n-bit result is fed into the generator. The generator uses a divisor of size n − k + 1 (4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division).**
- **The quotient of the division is discarded; the remainder (r2r1r0) is appended to the dataword to create the codeword.**
- **The decoder receives the codeword (possibly corrupted in transition). A copy of all n bits is fed to the checker, which is a replica of the generator.**
- **The remainder produced by the checker is a syndrome of n − k (3 here) bits, which is fed to the decision logic analyzer. The analyzer has a simple function.**

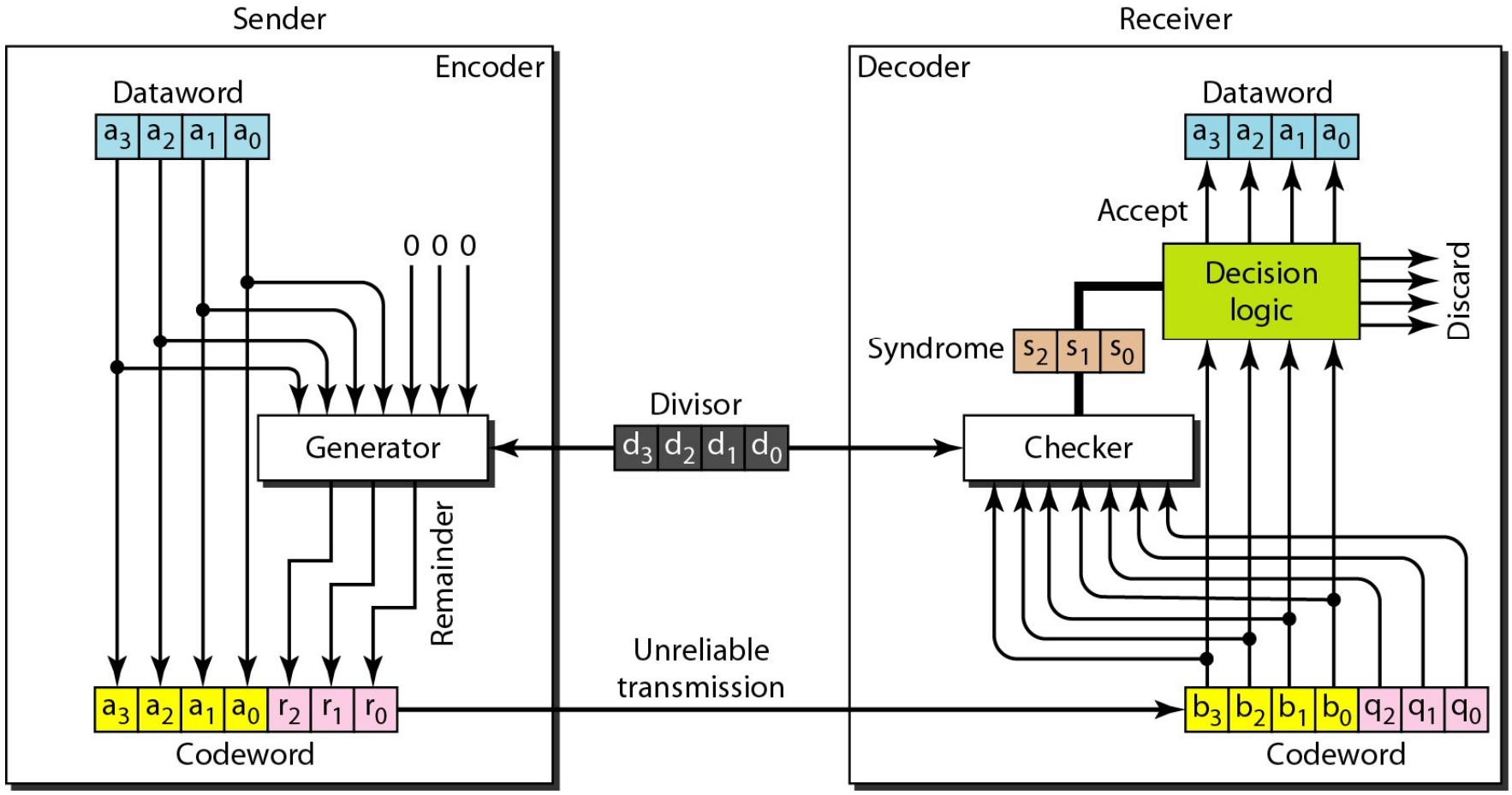
Cyclic Redundancy Check

- If the syndrome bits are all 0s, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded (error).

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

Table 10.6 *A CRC code with C(7, 4)*

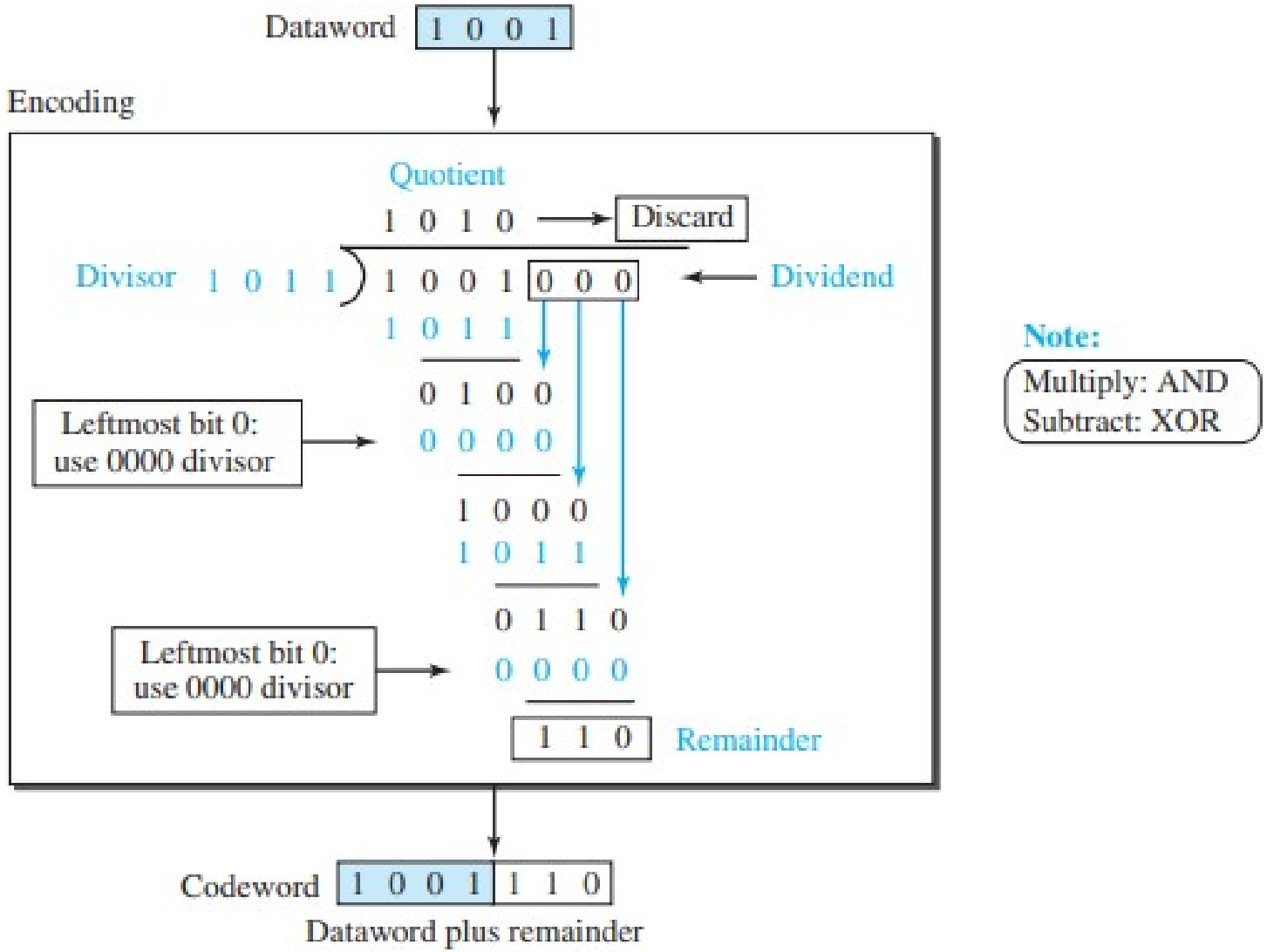
Figure 10.14 *CRC encoder and decoder*



Cyclic Redundancy Check

- **Encoder**
- **Encoder Let us take a closer look at the encoder. The encoder takes a dataword and augments it with n − k number of 0s. It then divides the augmented dataword by the divisor, as shown in Figure 10.6.**
- **The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. However, addition and subtraction in this case are the same; we use the XOR operation to do both.**
- **If the leftmost bit of the dividend (or the part used in each step) is 0, the step cannot use the regular divisor; we need to use an all-0s divisor.**
- **When there are no bits left to pull down, we have a result. The 3-bit remainder forms the check bits (r2, r1, and r0). They are appended to the dataword to create the codeword.**

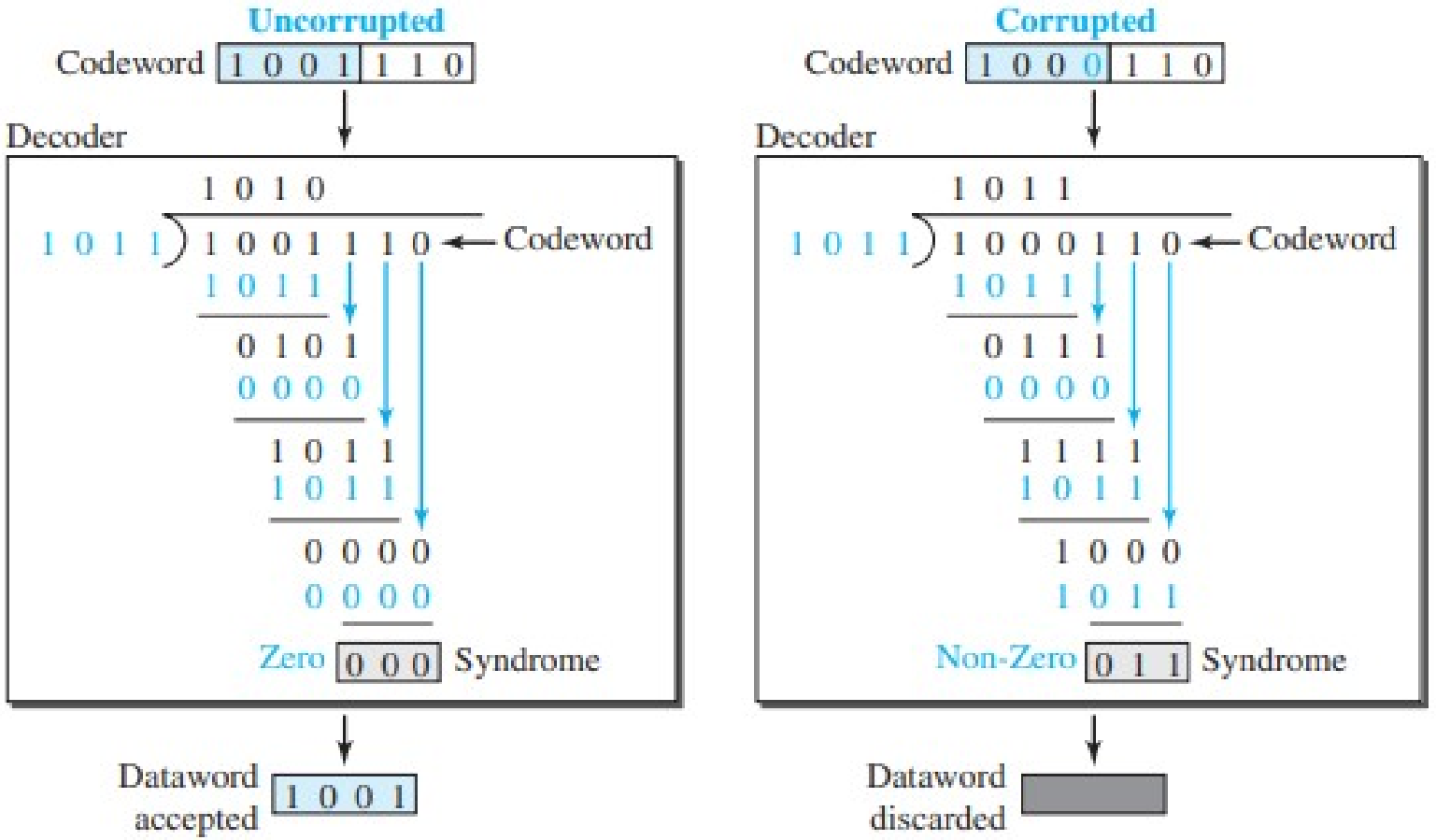
Figure 10.6 *Division in CRC encoder*



Cyclic Redundancy Check

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

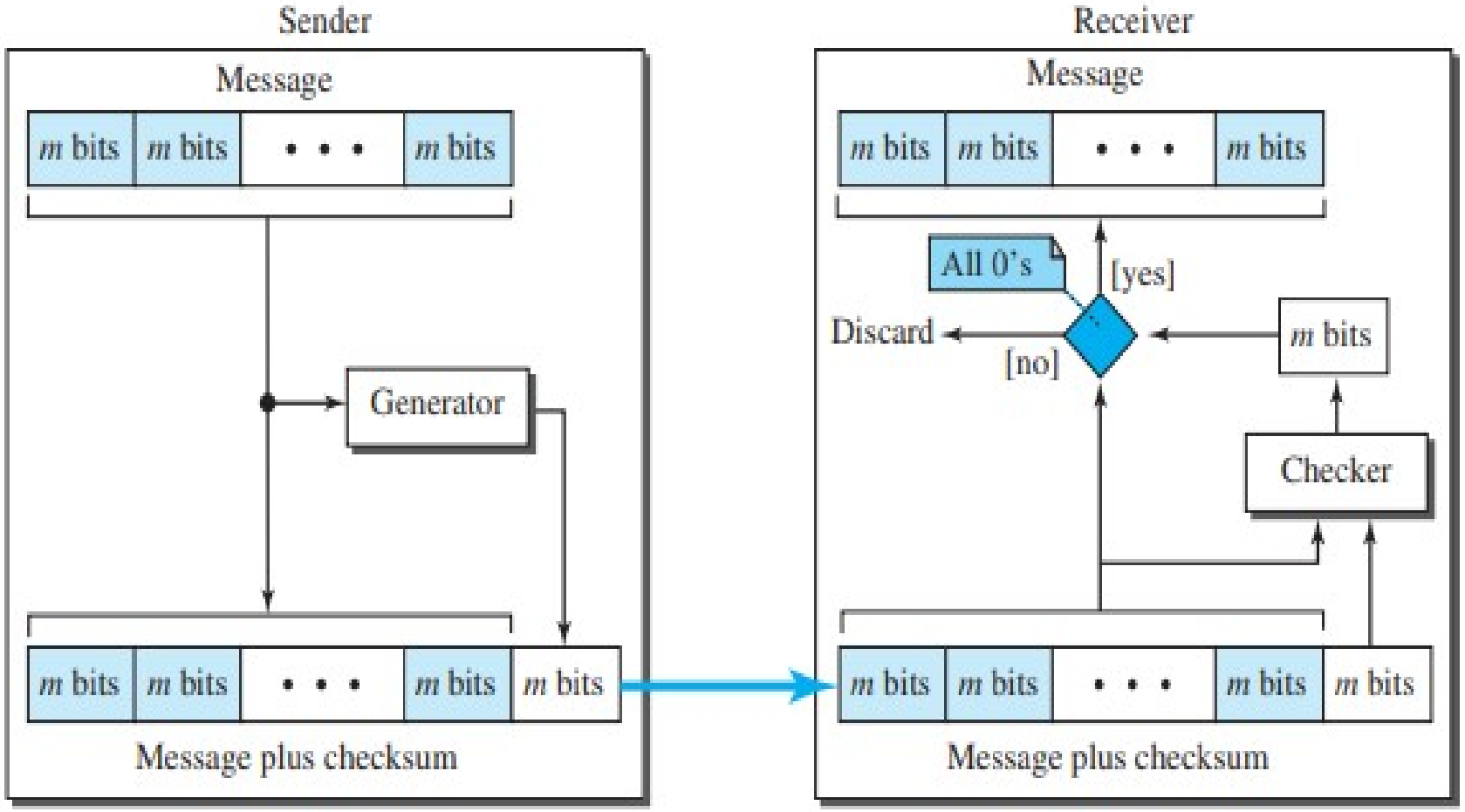
Figure 10.7 *Division in the CRC decoder for two cases*



Checksum

- **Checksum is an error-detecting technique that can be applied to a message of any length.**
- **In the Internet, the checksum technique is mostly used at the network and transport layer rather than the data-link layer.**
- **At the source, the message is first divided into m-bit units. The generator then creates an extra m-bit unit called the checksum, which is sent with the message.**
- **At the destination, the checker creates a new checksum from the combination of the message and sent checksum.**
- **If the new checksum is all 0s, the message is accepted; otherwise, the message is discarded (Figure 10.15).**

Figure 10.15 *Checksum*



Checksum

- Example 10.11
- **Suppose the message is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers.**
- **For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum.**
- **If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum.**
- **Otherwise, there is an error somewhere and the message is not accepted.**

Checksum

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

Example 10.12

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

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

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

Checksum

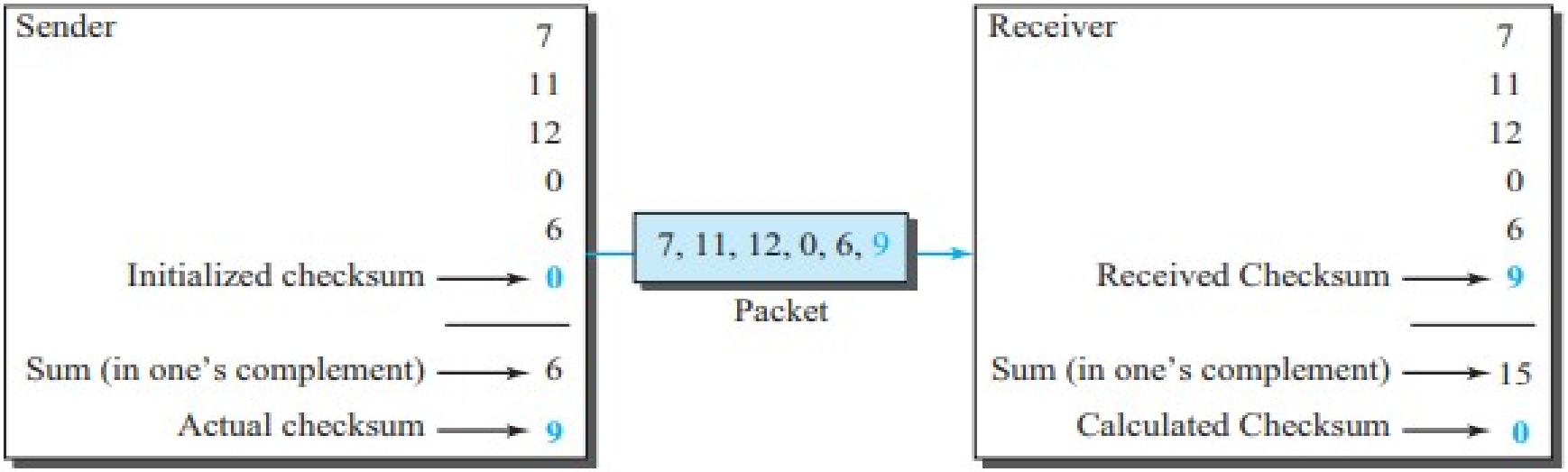
- **We can make the job of the receiver easier if we send the complement of the sum, the checksum.**
- **In one's complement arithmetic, the complement of a number is found by completing all bits (changing all 1s to 0s and all 0s to 1s). This is the same as subtracting the number from $2^m - 1$.**
-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

Checksum

Example 10.13

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

Figure 10.16 *Example 10.13*



Checksum

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

Table 10.5 *Procedure to calculate the traditional checksum*

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

Checksum

Figure 10.17 *Algorithm to calculate a traditional checksum*