Chapter 9: Introduction to Data-Link Layer

Outline

9.1 INTRODUCTION

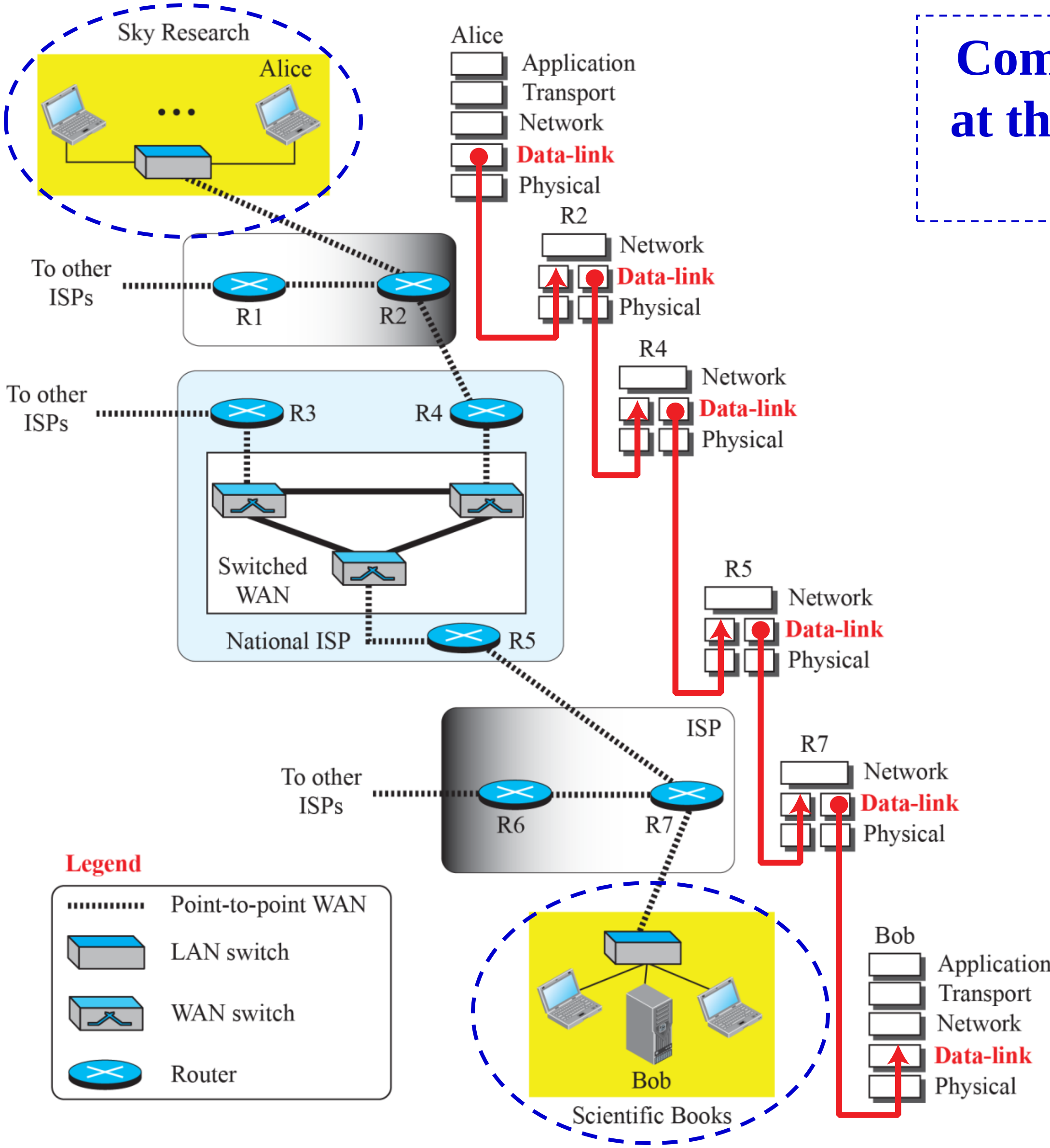
9.2 LINK-LAYER ADDRESSING

9-1 INTRODUCTION

The Internet is a combination of networks glued together by connecting devices (routers or switches).

If a packet is to travel from a host to another host, it needs to pass through these networks.

Figure 9.1: Communication at the data-link layer

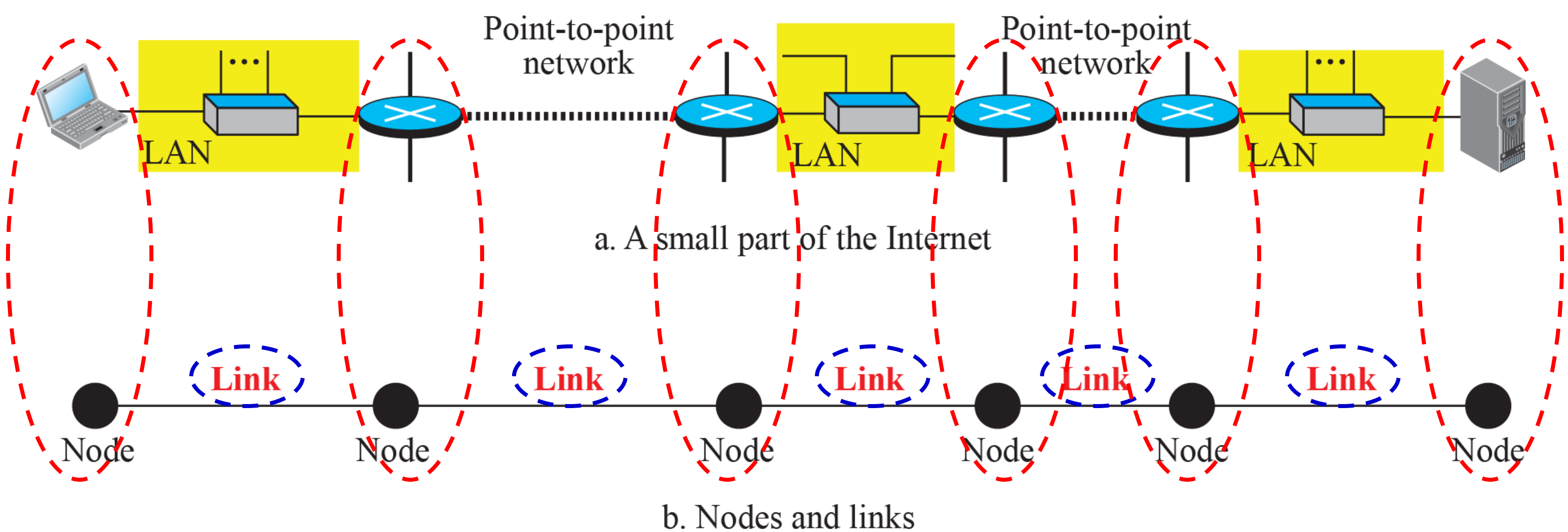


Communication at the Data-Link Layer

9.1.1 Nodes and Links

Communication at the data-link layer is <u>node-to-node</u>. A data unit from one point in the Internet needs to pass through many networks (LANs and WANs) to reach another point. These LANs and WANs are connected by routers.

It is customary to refer to the two <u>end hosts and the routers</u> as <u>nodes</u> and the <u>networks in between</u> as <u>links</u>.



9.1.2 Services

The <u>data-link layer</u> is located between the <u>physical</u> and the <u>network</u> layers. Services provided by the data-link layer include:

<u>Framing</u>: <u>Encapsulate</u> the <u>packet</u> received from the <u>network layer</u> in a <u>frame</u> before sending it to the next node. <u>Decapsulate</u> the packet from the frame received on the logical channel. <u>Different data-link layers have different formats for framing</u>.

<u>Flow Control</u>: If the <u>rate of produced frames</u> is <u>higher</u> than the <u>rate of processed frames</u>, frames at the receiving end need to be <u>buffered</u> while waiting to be processed. If buffer at receiving end is <u>full</u> ➔ <u>drop frames</u> / <u>feedback</u> to the sending node.

<u>Error Control</u>: <u>Frames are susceptible to error</u> in part due to electromagnetic signals being susceptible to error. The <u>error</u> needs to be <u>detected</u> and then either (i) <u>corrected</u> or (ii) <u>discarded and retransmitted</u> by the sending node.

<u>Congestion Control</u>: Some wide-area networks may use congestion control when a <u>link is congested with frames</u>. Mainly handled by <u>network</u> and <u>transport</u> layers.

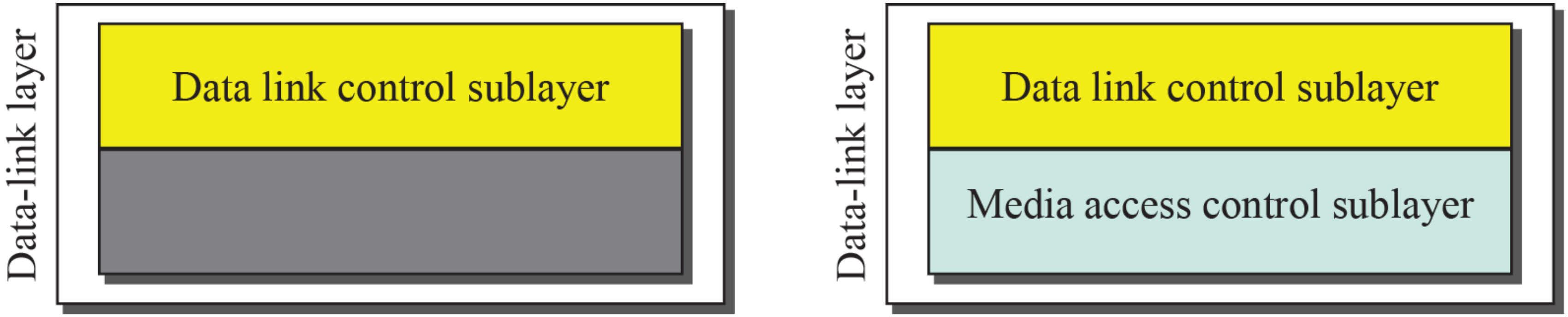
9.1.3 Two Categories of Links

While two nodes are physically connected by a transmission medium such as cable or air, the data-link layer controls how the medium is used. A data-link layer can use:

- ***the whole capacity of the medium***
  - ➔ ***the link is dedicated to the two devices***
  - ➔ ***point-to-point link***
- ***only part of the capacity of the medium***
  - ➔ ***the link is shared between several pairs of devices***
  - ➔ ***broadcast link***

9.1.4 Two Sublayers

To better understand the functionality of and the services provided by the link layer, the data-link layer is divided into two sublayers: data link control (DLC) and media access control (MAC).



Data-link layer of a point-to-point link

Data-link layer of a broadcast link

- ***The DLC sublayer deals with all issues common to both point-to-point and broadcast links.***
- ***The MAC sublayer deals only with issues specific to broadcast links.***

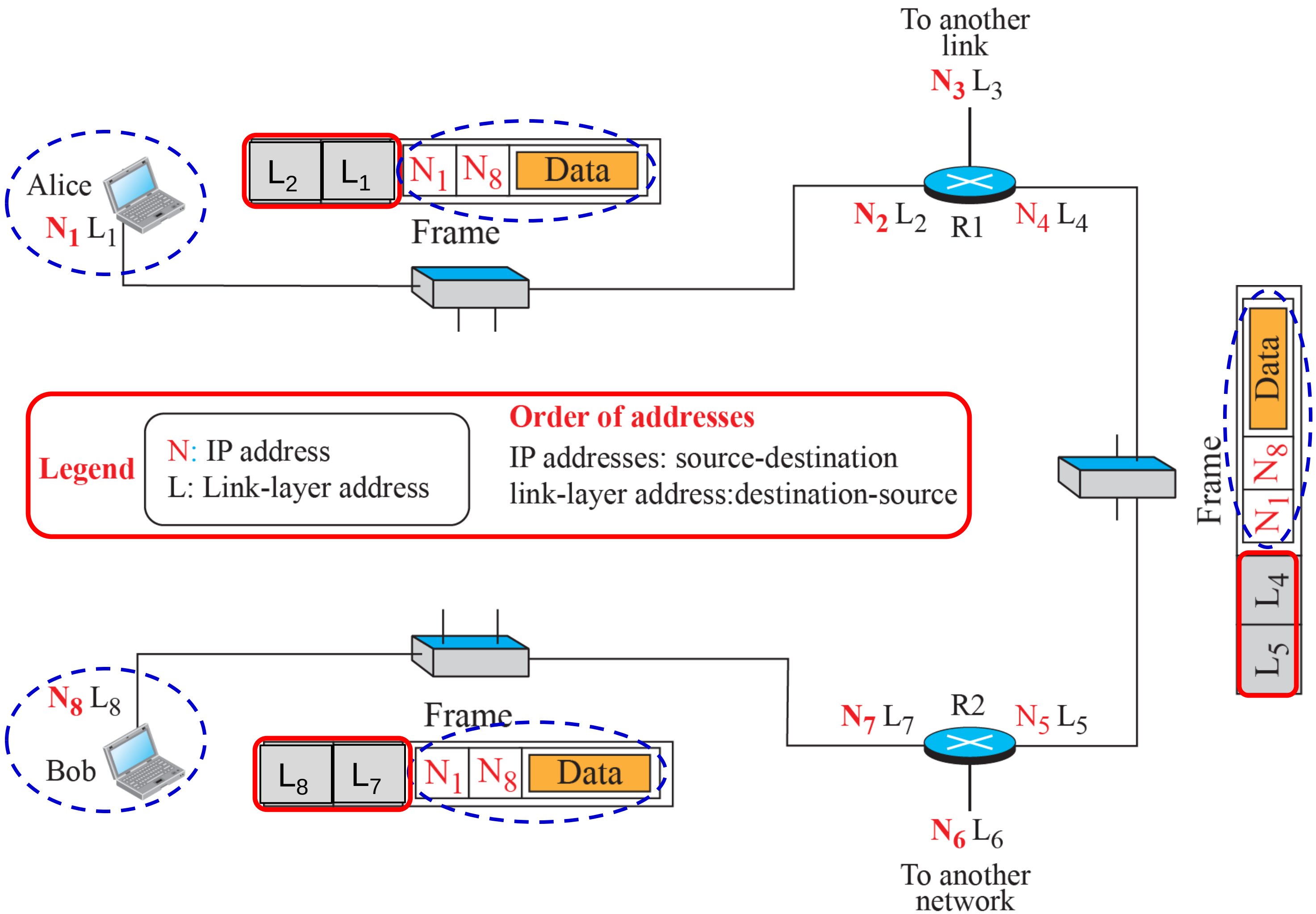
9-2 LINK-LAYER ADDRESSING

In a connectionless internetwork such as the Internet, a packet cannot reach its destination using only IP addresses. The source and destination IP addresses define the two ends but does not define which links the packet should pass through.

Each packet in the Internet, from the same source host to the same destination host, may take a different path.

Figure 9.5: IP addresses and link-layer addresses in a small internet

IP addresses in a packet should not be changed. Addressing mechanism in the data-link layer will encapsulate the packet from the network layer in a frame and add data-link addresses (MAC address) to the frame header. These addresses are changed every time the frame moves from one link to another.



9.2.1 Three Types of addresses

The link-layer addresses in Ethernet are 48 bits (6 bytes) that are presented as 12 hexadecimal digits separated by colons. Some link-layer protocols define three types of addresses:

- ***unicast:*** ***one-to-one communication***
  (The second digit needs to be an odd number, e.g., **A3:34:45:11:92:F1)**
- ***multicast:*** ***one-to-many communication***
  (The second digit needs to be an even number, e.g., **A2:34:45:11:92:F1**)
- ***broadcast:*** ***one-to-all communication***
  (The broadcast link-layer addresses in Ethernet are all 1s, e.g., **FF:FF:FF:FF:FF:FF)**

9.2.2 Address Resolution Protocol

When a node has an IP packet to send to another node in a link, it has the IP address of the receiving node. However, as discussed, the IP address of the receiving node is not helpful in moving a frame through a link. The link-layer address of the next node is required.

The Address Resolution Protocol (ARP), an auxiliary protocol defined in the network layer, accepts an IP address from the IP protocol, maps the IP address to the corresponding link-layer address and passes the link-layer address to the data-link layer.

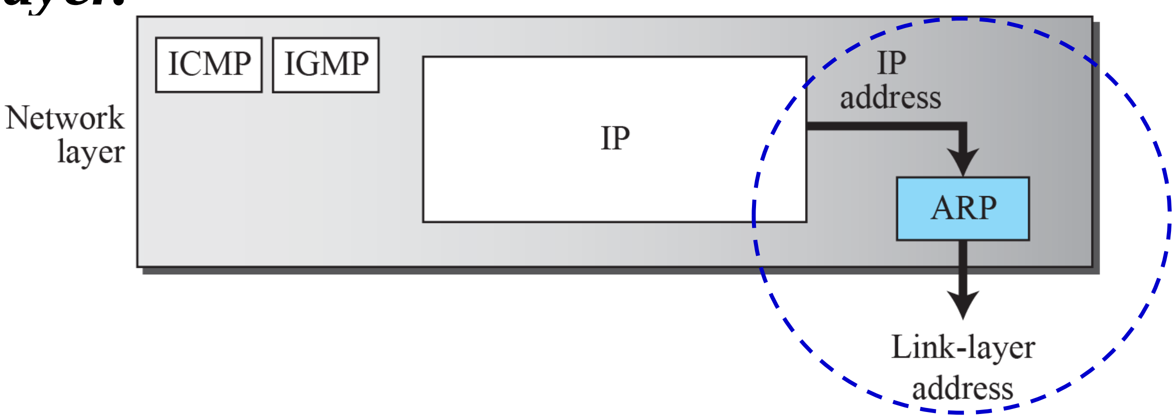
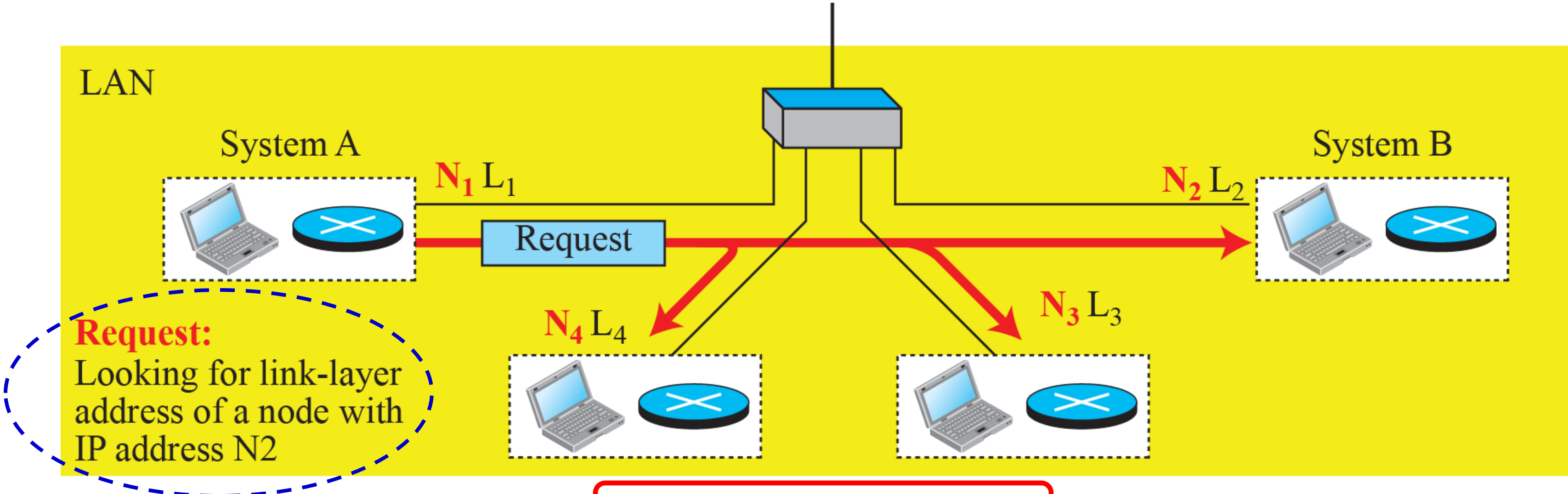
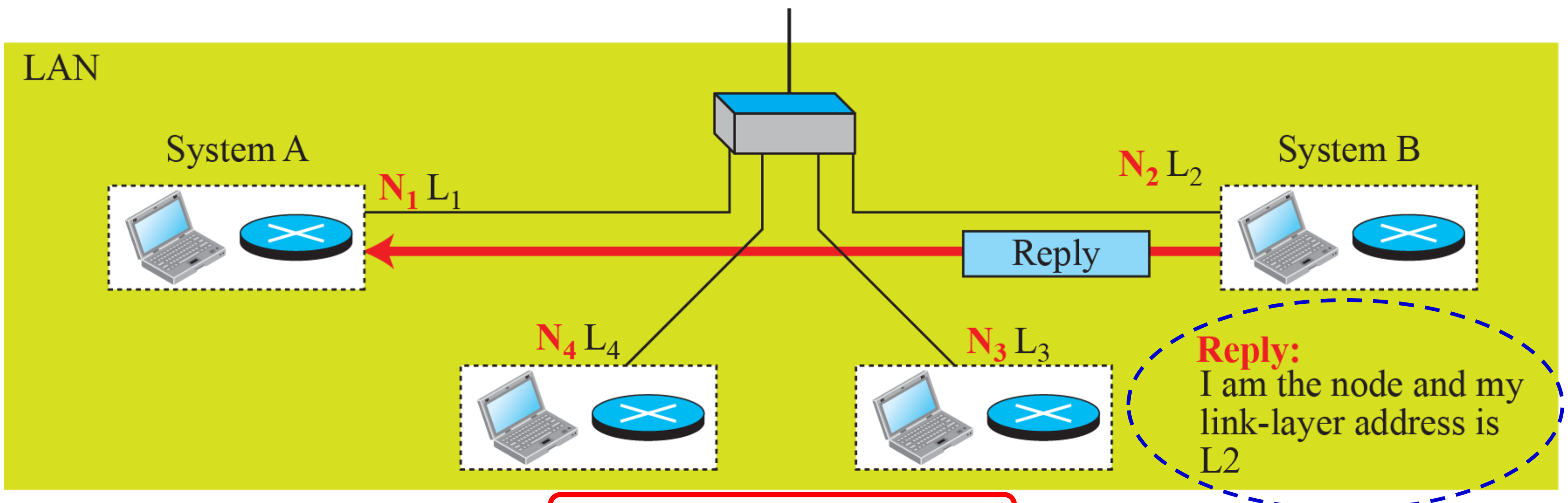


Figure 9.7: ARP operation



a. ARP request is broadcast



b. ARP reply is unicast

Chapter 10: Error Detection and Correction

Outline

10.1 INTRODUCTION

10.2 BLOCK CODING

10.3 CYCLIC CODES

10.4 CHECKSUM

10-1 INTRODUCTION

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 number of bits affected depends on the data rate and the duration of noise.

10.1.1 Types of Errors

The term <u>single-bit error</u> 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 <u>burst error</u> means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1.

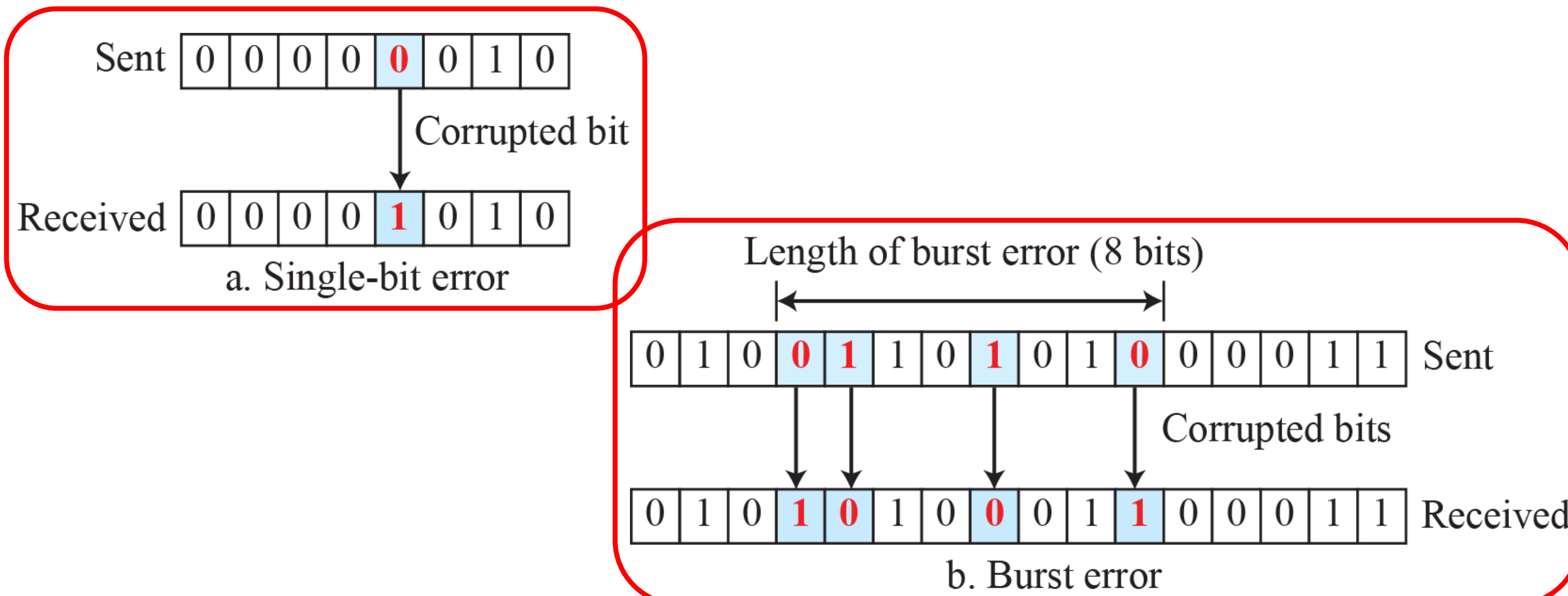


Figure 10.1: Single-bit and burst error

A burst error is more likely to occur than a single-bit error because the duration of the noise is normally longer than the duration of 1 bit (data rates are in orders of Mbps, Gbps and increasing).

For example:

- ***If data is being sent at 1 kbps, a noise of 1/100 sec can affect 10 bits.***
- ***If data is being at 1 Mbps, the same noise of 1/100 sec can affect 10,000 bits.***

10.1.2 Redundancy

The central concept in <u>detecting or correcting errors</u> is <u>redundancy</u>.

To be able to detect or correct errors, we need to send some <u>extra bits</u> with our data. These redundant bits are <u>added by the sender</u> and <u>removed by the receiver</u>.

The presence of the redundant bits allows the <u>receiver</u> to <u>detect or correct corrupted bits</u>.

10.1.3 Detection versus Correction

The correction of errors is more difficult than the detection of errors.

In error detection, we are only trying to determine if any error has occurred. We are not interested in the number of corrupted bits. In error detection, whether single-bit errors or burst errors occurred may be regarded in the same way.

In error correction, we need to know the exact number of bits that are corrupted and, more importantly, the location of the bit errors in the message. Consider the receiver's difficulty in finding 10 errors in a data unit of 1000 bits ➔ $\binom{1000}{10} = 263 \times 10^{21}$.

Recall $\binom{n}{k} = \left(\frac{n!}{k!(n-k)!}\right)$

10.1.4 Coding

Redundancy is achieved through various <u>channel coding</u> schemes. The <u>sender</u> adds <u>redundant bits</u> through a process that creates a <u>relationship</u> between the redundant bits and the actual data bits. The <u>receiver</u> checks the <u>relationships</u> between the <u>two sets</u> of bits to <u>detect errors</u>.

The <u>code rate</u> (proportion of data stream that is useful), and the <u>robustness</u> (to channel impairments) are important factors in any coding scheme.

There is another aspect to coding theory, called <u>source coding</u>, that attempts to compress the data in order to transmit it more efficiently. Source coding (data compression) is not covered in this course.

10-2 BLOCK CODING

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

We will discuss later as to how the extra r bits are chosen or calculated.

10.2.1 Error Detection

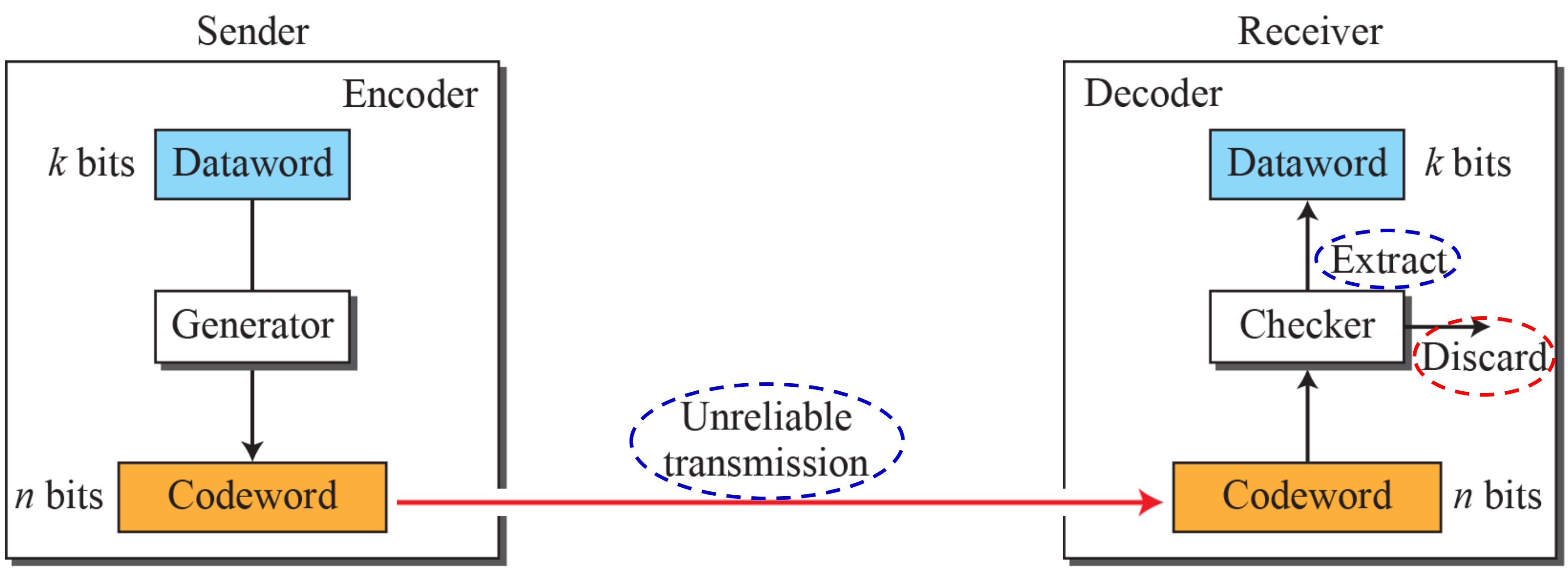
Since n > k, the number of possible codewords (2^n) is larger than the number of possible datawords (2^k). In block coding, the same dataword is always encoded as the same codeword, i.e., there are (2^n – 2^k) unused (invalid) codewords.

We exploit the existence of these invalid codewords to detect errors.

Figure 10.2: **Process of error detection in block coding**

The receiver can detect a change in the original codeword if the following two conditions are met:

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



Note that if the codeword is corrupted during transmission but the received codeword still matches a valid codeword, the error remains undetected.

Problem

Let us assume that k = 2 and n = 3. The list of datawords and codewords is as shown. Assume that the sender encodes the dataword **01** as **011** and sends it to the receiver, determine the dataword extracted at the receiver for each of the following:

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

a) The receiver receives **011**.

b) The receiver receives **111**.

c) The receiver receives **000**.

Hamming Distance

The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits. The hamming distance between two words x and y is denoted by d(x, y).

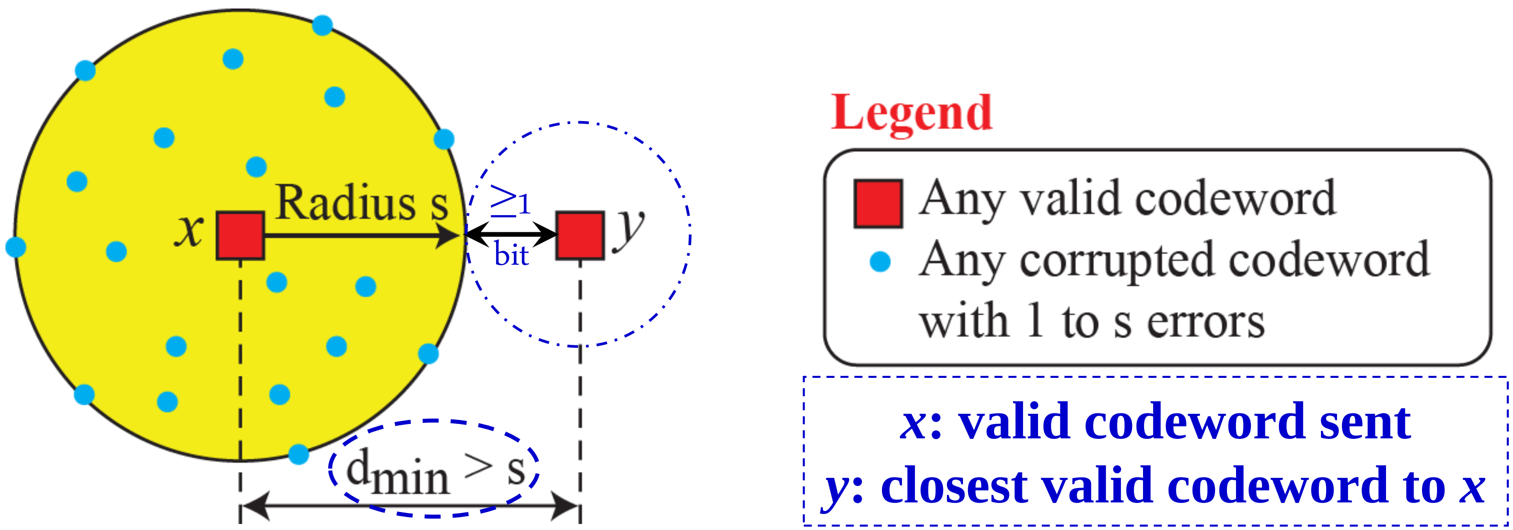
The significance of Hamming distance in error detection is that it determines the number of bits that are corrupted during transmission between the received codeword, R_C, and the transmitted codeword, T_C.

Idea: the codeword has not been corrupted during transmission if $d(R_C, T_C) = 0$.

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

We can obtain the Hamming distance, d(x, y), by applying the XOR operation (⊕) on x and y and counting the number of 1s in the result.

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



Problem

1) Determine the Hamming distance, d(x, y), of the following:

a) d(000, 011) =

b) d(10101, 11110) =

c) d_{min}(000, 011, 101, 110) =

2) For a given code scheme with $d_{min} = 4$, how many errors are guaranteed to be detected?

Linear Block Codes

An informal definition of a Linear Block Code (LBC) is a code in which the <u>XOR of two valid codewords creates</u> another <u>valid codeword</u>. E.g., the code scheme 000, 011, 101 and 110 is a LBC since

$$\boxed{011 \oplus 101 = 110;\ 011 \oplus 110 = 101;\ 101 \oplus 110 = 011}$$

The d_{min} for a LBC is the <u>number of 1s</u> in the <u>nonzero valid codeword</u> with the <u>smallest number of 1s</u>. In the above code scheme, the number of 1s in the nonzero codewords are 2, 2 and 2. Hence, d_{min} = 2.

Parity-Check Code

In parity-check code, a LBC, a <u>k-bit dataword</u> is changed to an <u>n-bit codeword</u> where n = k + 1. The extra bit, called the <u>parity bit</u>, is selected to make the total number of 1s in the codeword <u>even or odd</u>.

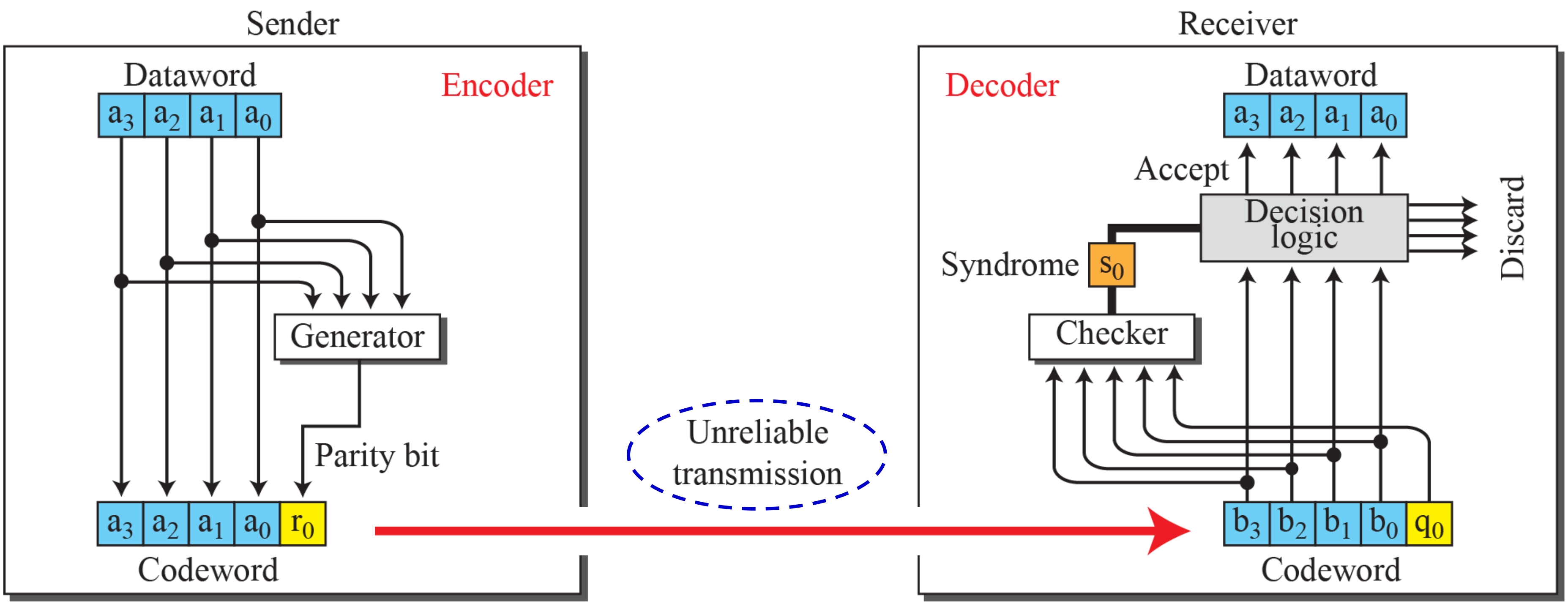
Example of an <u>even parity-check code</u> with n=5, k=4; the d_{min} is 2, i.e., the code is a <u>single-bit</u> error-detecting code.

Parity Bit

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**

Figure 10.4: Encoder and decoder for <u>even parity-check code</u>

The calculation of the parity bit is performed using <u>modulo-2 arithmetic</u> (Appendix E).



$$r_0 = a_3 + a_2 + a_1 + a_0$$

5-bit codeword $a_3a_2a_1a_0r_0$ for even parity-check code has <u>even number</u> of 1s.

$$s_0 = b_3 + b_2 + b_1 + b_0 + q_0$$

If syndrome s_0 = 0, there is <u>no error</u>; dataword is <u>accepted</u>.
If syndrome s_0 = 1, an <u>error</u> occurred; dataword is <u>discarded</u>.

Problem

Assume the sender sends the dataword **1011**. The parity-check codeword ($a_3a_2a_1a_0r_0$) created from this dataword is **10111**, which is sent to the receiver. We examine five cases:

a) No error occurs; the received codeword is **10111**.

b) One single-bit error changes a_1; the received codeword is **10011**.

c) One single-bit error changes r_0; the received codeword is **10110**.

d) An error changes r_0 and a second error changes a_3; the received codeword is **00110**.

e) Three bits, a_3, a_2, a_1 are changed by errors; the received codeword is **01011**.

10-3 CYCLIC CODES

Cyclic codes are special linear block codes with one extra property: in a cyclic code, if a codeword is cyclically shifted, the result is another codeword.

For example, if 1011000 is a codeword and we cyclically left-shift, then the result, 0110001, is also a codeword.

10.3.1 Cyclic Redundancy Check

We can create cyclic codes to correct errors. In this section, we simply discuss a subset of cyclic codes called the <u>cyclic redundancy check</u> (CRC), which is widely used in networks such as LANs and WANs.

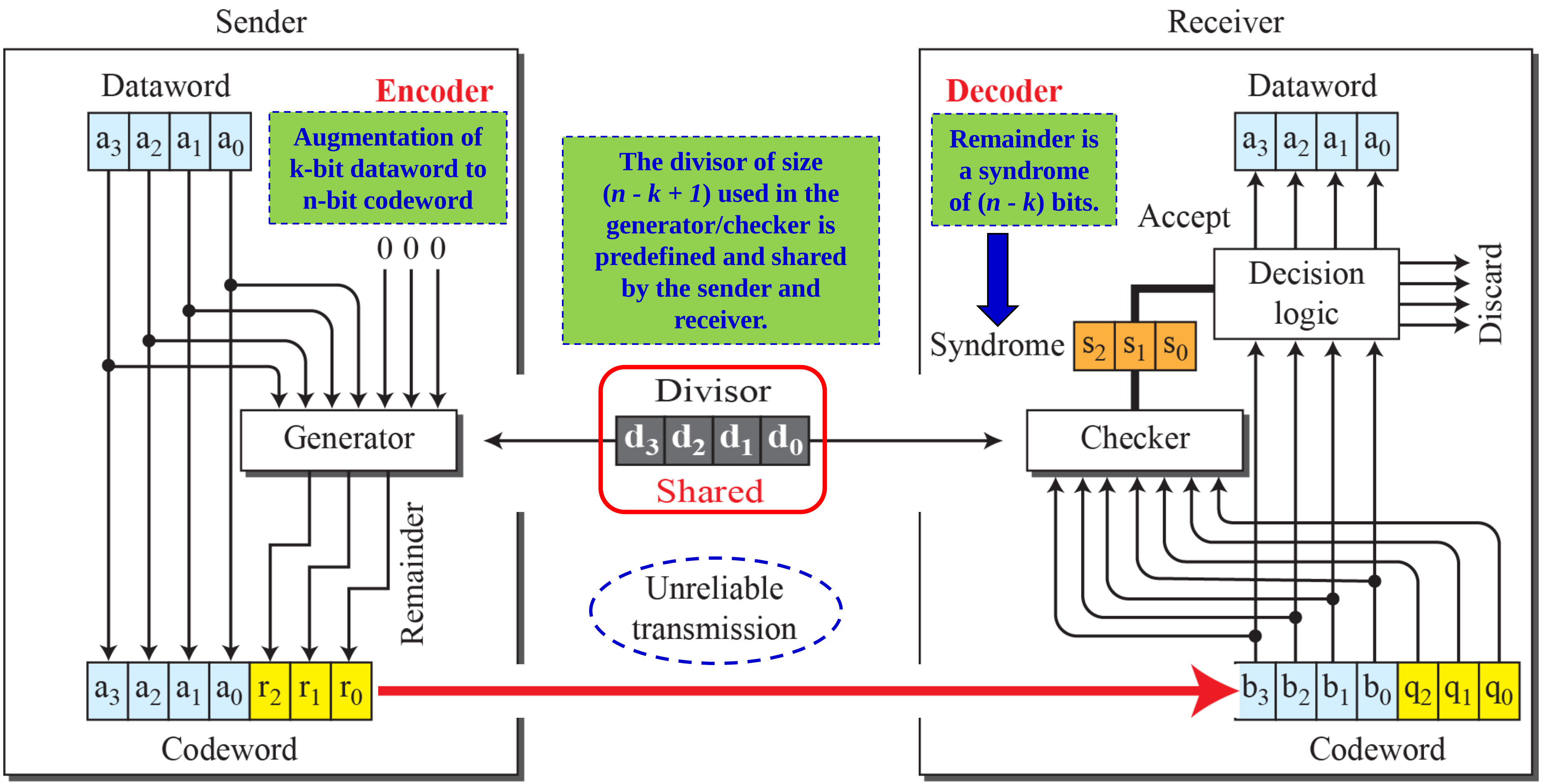
Example of a CRC code with n=7, k=4:

Cyclic Codes

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

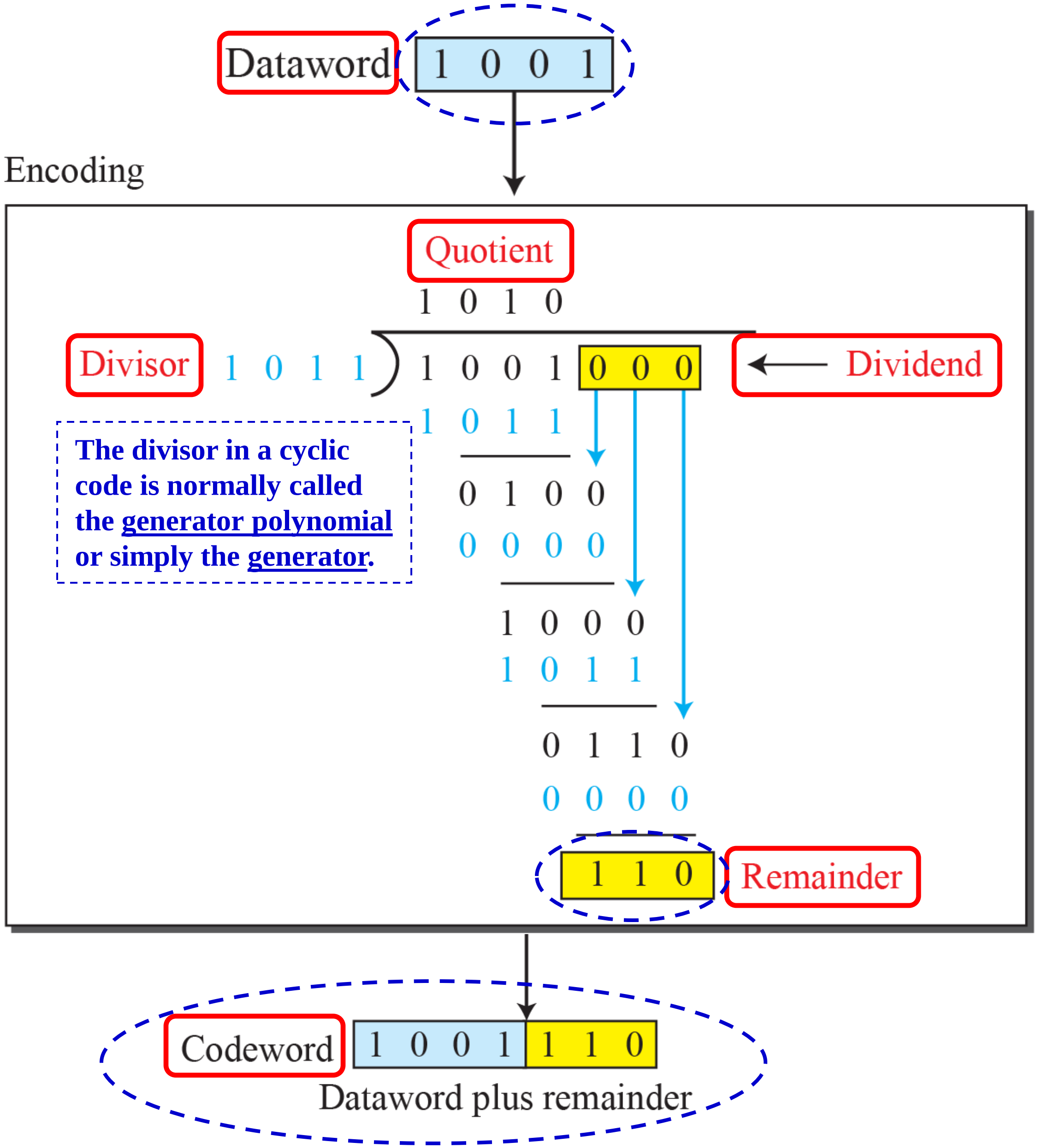
Figure 10.5: *CRC encoder and decoder*

Encoding/Decoding of an n-bit CRC codeword with a k-bit dataword.



If the syndrome bits are all 0s, there is no error; dataword $b_3b_2b_1b_0$ is accepted. Otherwise, an error occurred, $b_3b_2b_1b_0$ is discarded.

Figure 10.6: Division in CRC encoder



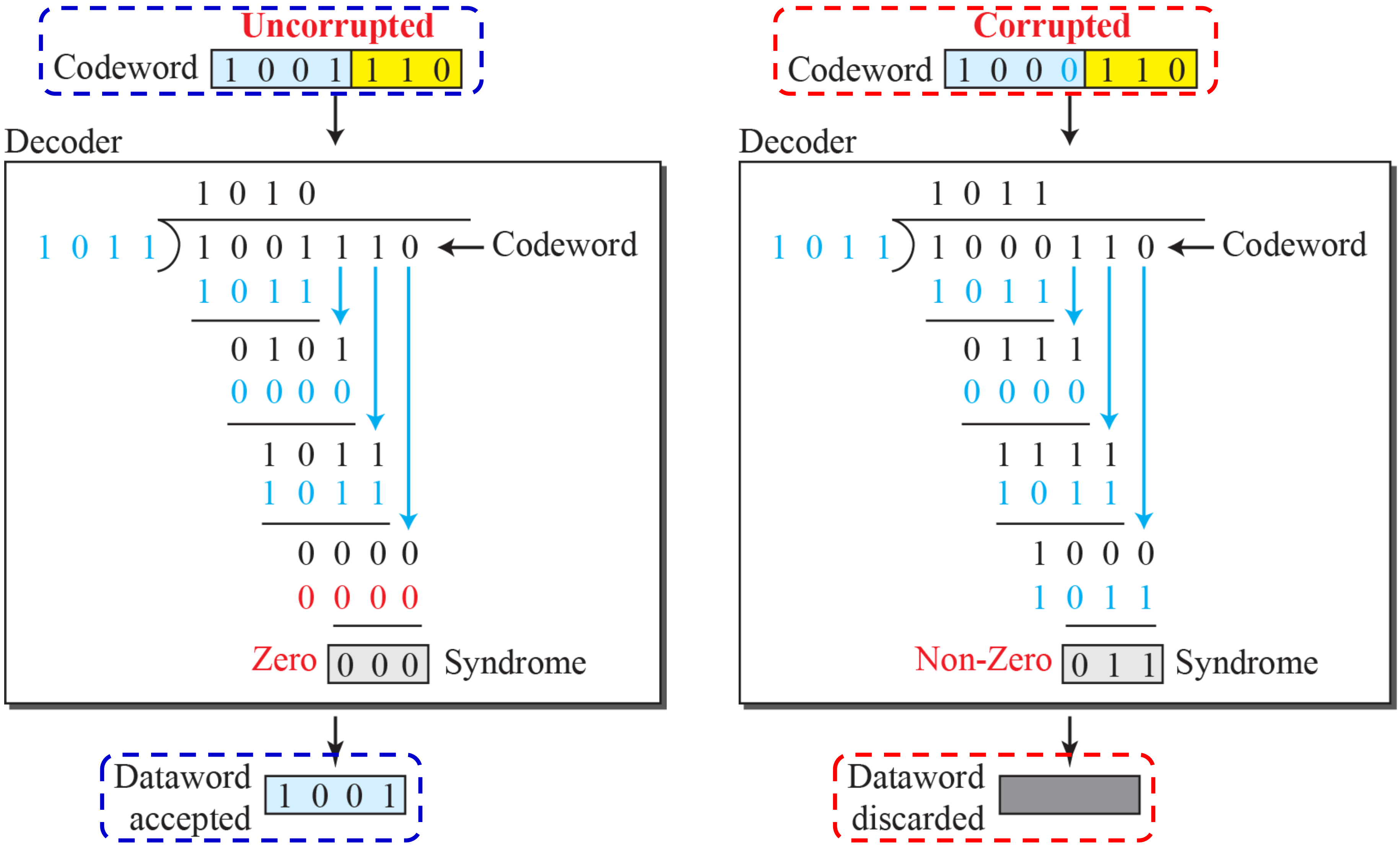
Note:

Multiply: AND
Subtract: XOR

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Figure 10.7: Division in the CRC decoder for two cases

Encoder sent CRC codeword 1001110.

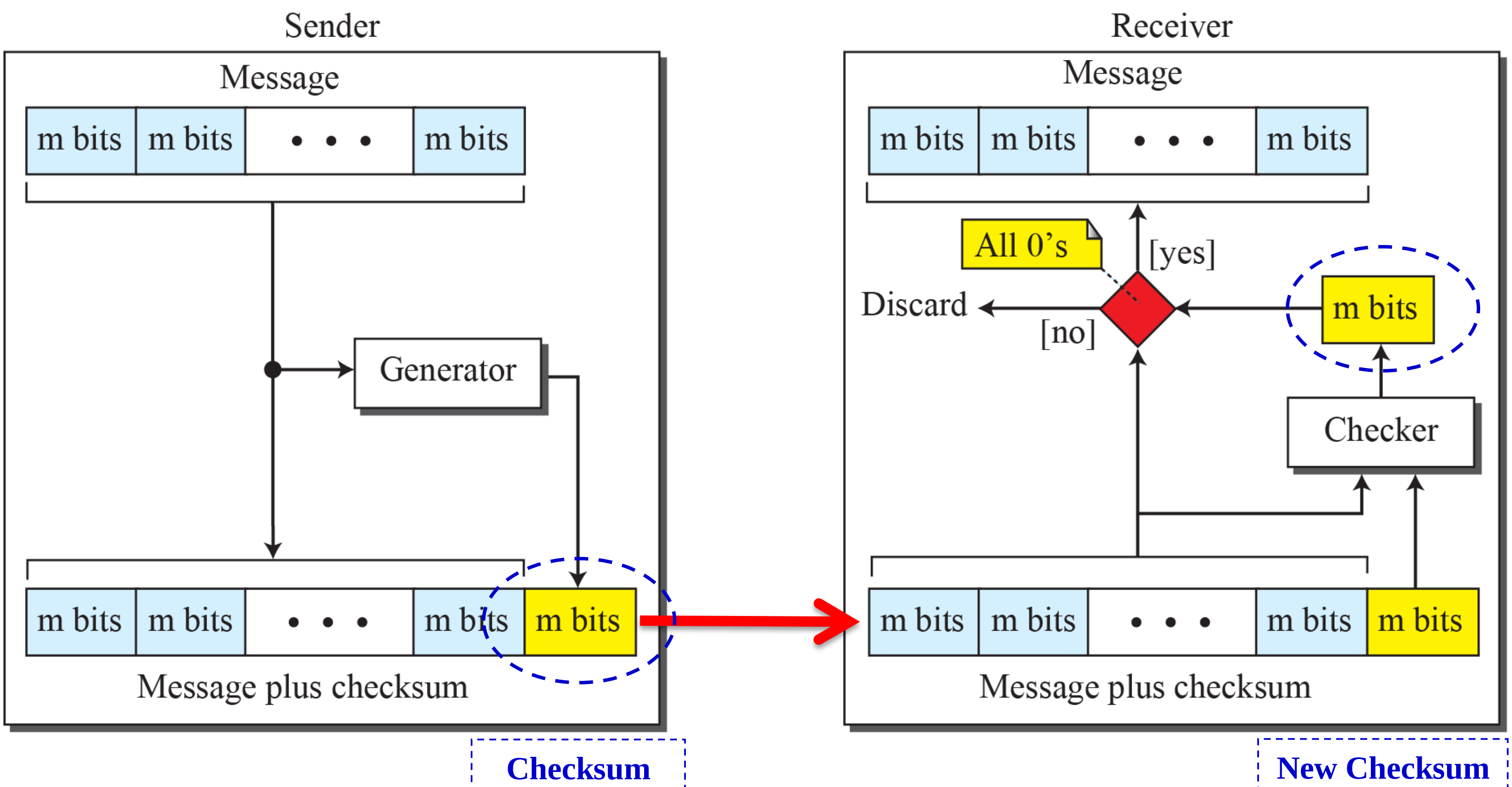


10-4 CHECKSUM

Checksum is an error-detecting technique that can be applied to a message of any length. In the Internet, the checksum technique is mostly used at the network and transport layer rather than the data-link layer.

Figure 10.15: Checksum

At the source, the <u>message</u> is first divided into <u>m-bit units</u>. The generator then creates an <u>extra m-bit unit</u> called the <u>checksum</u>, which is sent with the message. At the destination, the checker creates a <u>new checksum</u> from the combination of the message and sent checksum. If the new checksum is <u>all 0's</u>, the <u>message</u> is <u>accepted</u>; otherwise, the message is discarded.



Example

The idea of the checksum is simple. Let's show this using a simple example:

Suppose the message is a list of five 4-bit numbers (7, 11, 12, 0, 6). In addition to sending these numbers, we send the sum of the numbers (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, it discards the sum and accepts the five numbers. Otherwise, an error occurred and the message not accepted.

However, the checksum $(36)_{10}$ in binary is $(100100)_2$, a 6-bit number. A solution is to use **one's complement arithmetic**: To change it to a 4-bit number we add the leftmost bits to the right four bits (end-around carry) as:

$$\mathbf{100100_2 \Rightarrow 10_2 + 0100_2 = 0110_2 = 6_{10}}$$

Hence, instead of sending $(36)_{10}$ as the sum, we send $(6)_{10}$ as the sum, i.e., (7, 11, 12, 0, 6, **6**). The receiver adds the first five numbers in one's complement arithmetic and compares the result with the sum.

Table 10.5: Procedure to calculate the checksum

Traditionally, the Internet uses a <u>16-bit checksum</u> and the sender and receiver follow the steps depicted below:

Sender	*Receiver*
1. The message is divided into 16-bit words.	1. The message and the checksum is 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.

Example

As an example, for the numbers (7, 11, 12, 0, 6), 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**, i.e., 15 – 6.

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 sum is complemented to obtain the calculated checksum. As the calculated checksum value is 0, the message is accepted.