CN (IT-3001)

Link Layer: DLC Sublayer

Prof. Amit Jha

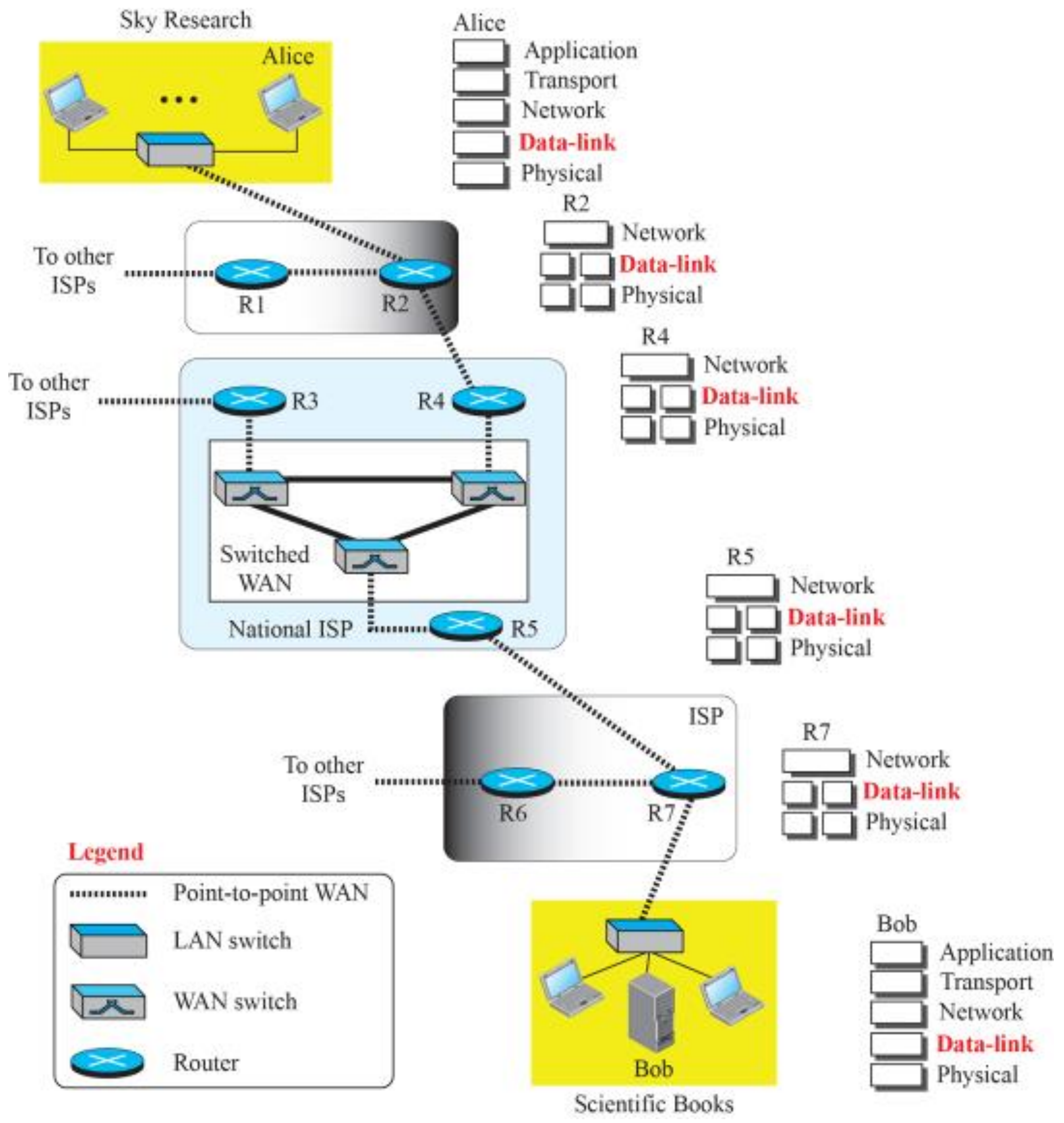
School of Electronics Engineering (SOEE)

KIIT Deemed to be University

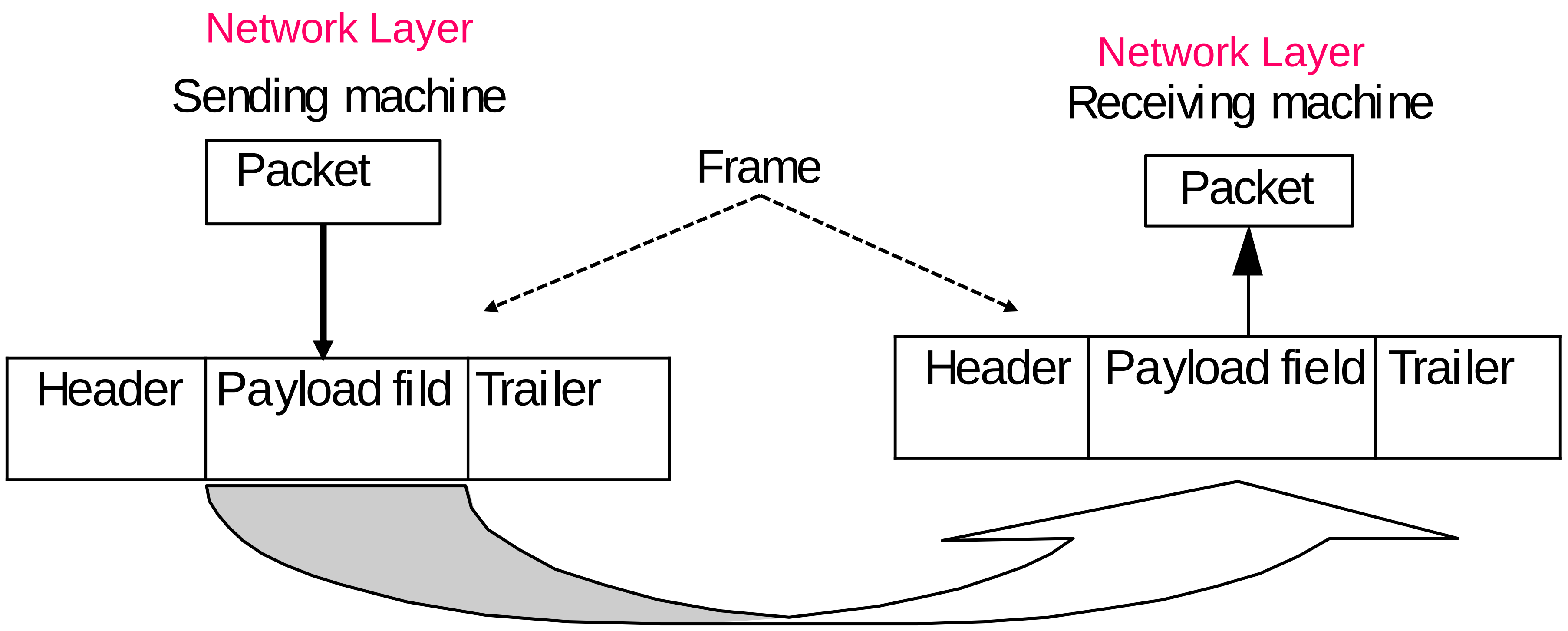
Objective

- To understand the introduction and services of Data Link Layer
- Data Link Control
  - Error Detection and Correction
  - Framing
- Multiple Access
- Ethernet Frame Format and Addressing
- GIGABIT Ethernet
- Link Layer Switching and VLANs

Data Link Layer: An Overview



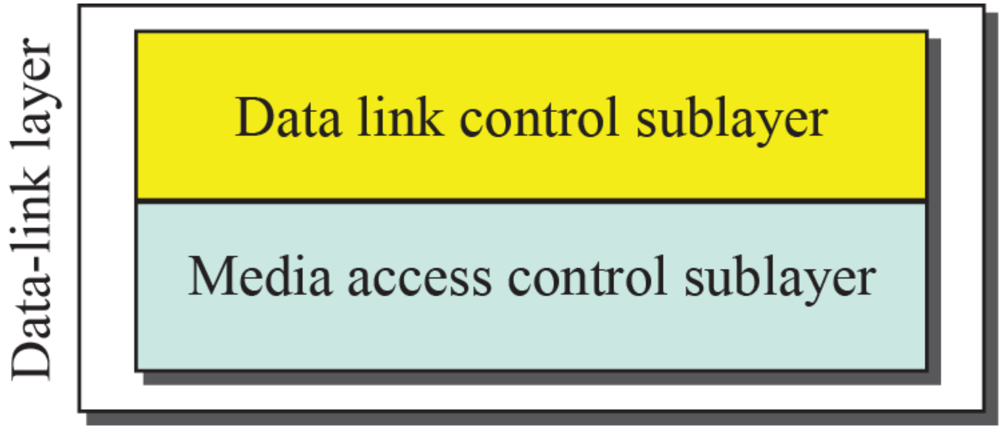
"Packet" and "Frame" relationship



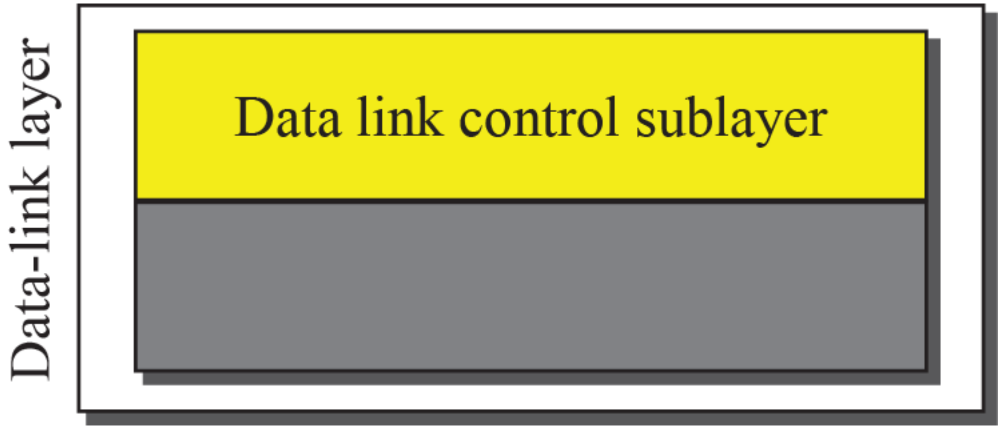
In some cases, functions of error control and flow control are allocated in transport or other upper layer protocols and not in the DLL, but principles are pretty much the same.

Two Sublayers of the Data-Link-Layer

- The data link layer is divided into two sublayers as shown below.
  1. Data Link Control (DLC) sublayer
  2. Media Access Control (MAC) Sublayer

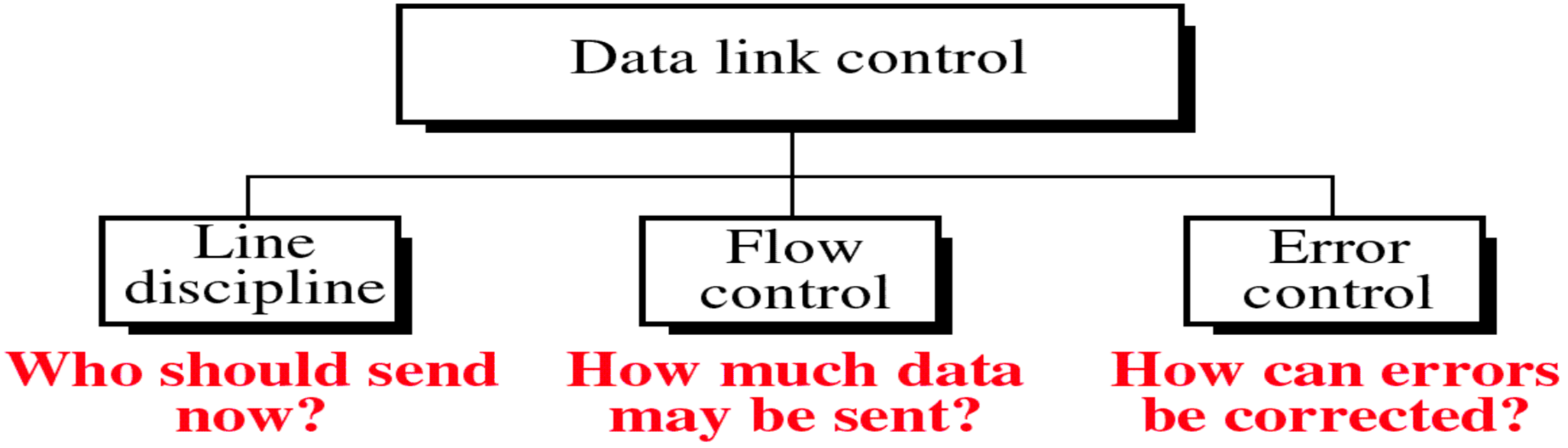


a. Data-link layer of a broadcast link



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

Responsibilities of Data Link Layer



- Specific responsibilities of the data link layer include ***framing, addressing***, ***flow control***, ***error control***, and ***media access control***.
- It provides service interface to the network layer.
- The data link layer adds a header to the frame to define the addresses of the sender and receiver of the frame.

Responsibilities of DLL

- ***Framing:***

- Data link layer deals data in chunks generally called ***Frames.***
- Size of frame is typically a few hundreds/thousands of bytes.
- The data link layer is concerned with local delivery of frames between devices on the same LAN.
- It creates/recognizes frame boundaries i.e., start and stop of the frame in order to distinguish between them.

Responsibilities of DLC and MAC sublayer of DLL

DLC sublayer is responsible for →

1. *Framing,*
2. *flow and Error control, and*
3. *error detection and correction.*

MAC sublayer is responsible for →

1. *Channel access*
2. *Media control*

- ***The several DLC protocol includes***
  - ***HDLC***
  - ***ATM***
  - ***Frame Relay***

Framing

- Data link layer converts stream of bits (from physical layer) into frames.
- Each frame is made distinguishable from one another by appending source and destination address.
- Although, the whole message can be packed in one frame, that is not normally done. This is because, a very large frame will not make flow and error control very efficient.
- For e.g., even for a single bit error, we need to retransmit the complete frame.

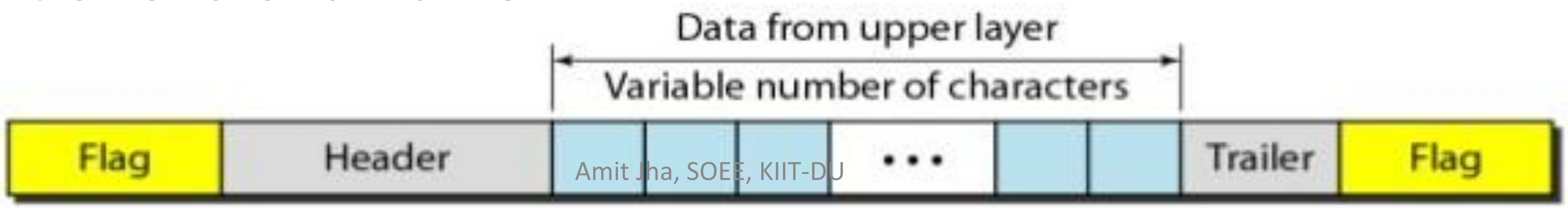
Frame Size

- Size of the frame can be either fixed or variable.
- **Fixed-Size Framing:** In fixed-size framing, there is no need for defining the boundaries of the frames; the size itself can be used as a delimiter. An example of this type of framing is the **ATM wide-area network**, which uses frames of fixed size called cells.
- **Variable-Size framing:** It is the most important as it is widely used in **local-area networks**. In variable-size framing, we need a way to defined boundaries, i.e., beginning and end of the frames.
  - Historically, two approaches were used for this purpose:

  1) Character-oriented approach   2) Bit-oriented approach

Variable-Size Framing: Character-Oriented Protocol

- In character oriented protocol, data to be carried are 8-bit characters from a coding system such as ASCII.
- The header which consists of source and destination addresses and other control information, and the trailer, which carries error detection and correction redundant bits, are also multiple of 8 bits.
- To separate one frame from the next, an 8-bit(1-byte) flag is added at the beginning and the end of the frame.
- The flag composed of protocol dependent special characters, signals that start or end of a frame.



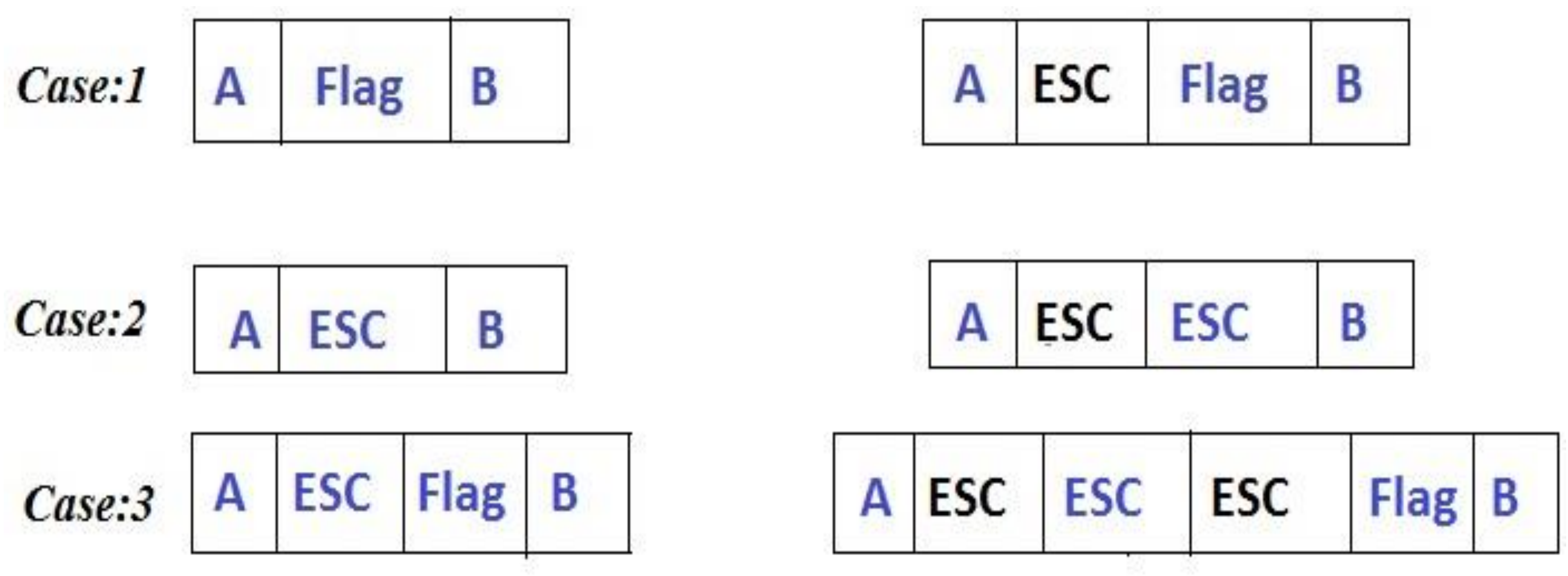
- **A challenge:** Flag can be any thing which is not used in the data to be sent using DLL. This is well when we use only text as a data to be sent. But in general, we are sending audio, video, images, etc. as data which can have the same pattern as used for flag.

Solution: To fix this problem, we use byte stuffing (or character stuffing).

- **Byte stuffing:** The data section is stuffed with an extra byte, called escape character (ESC), whenever there is a character with the same pattern as flag in the data. ESC character has a predefined bit pattern.

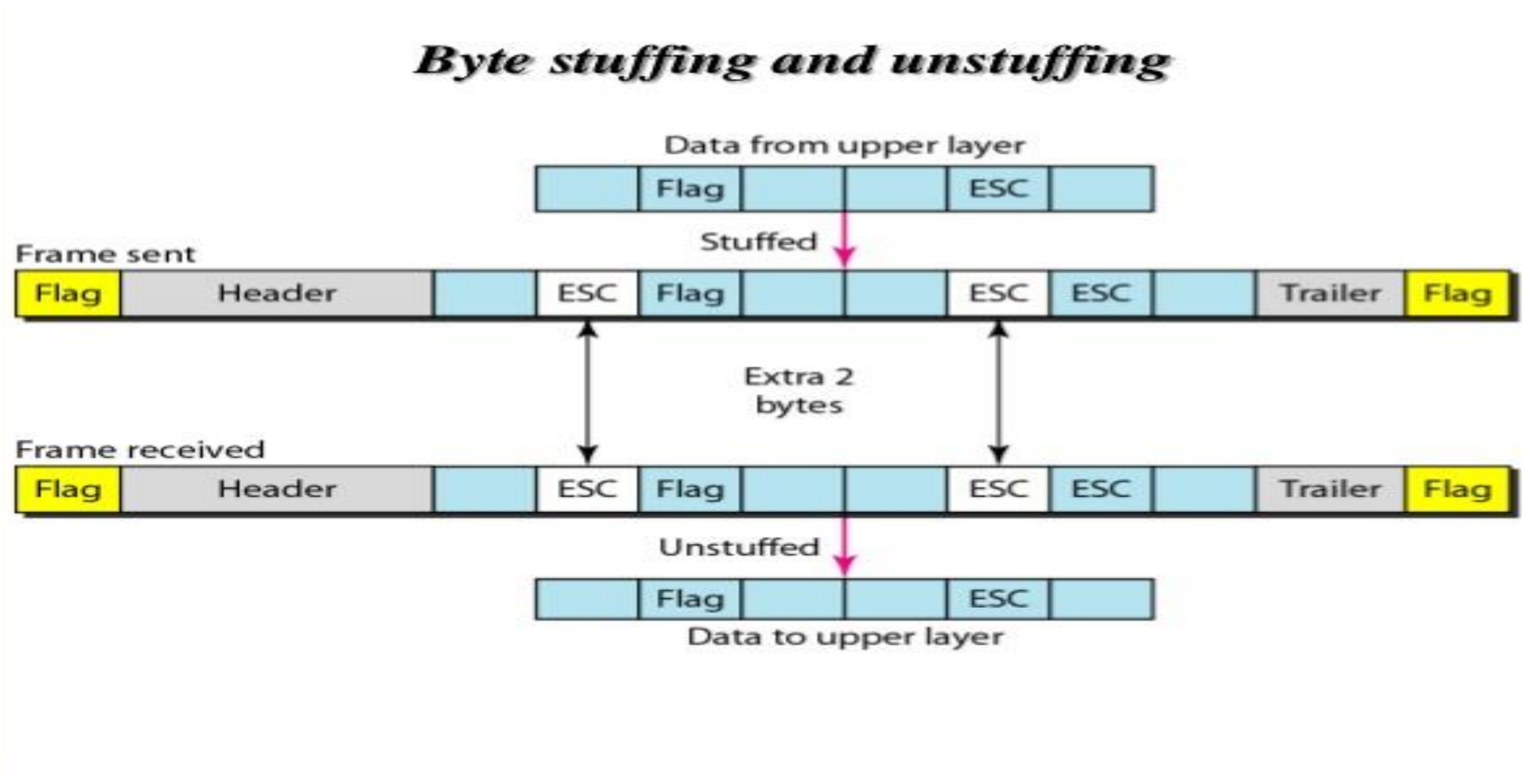
Whenever the receiver encounters the ESC character, it removes it from the data section and treats the next character as data, not a delimiting flag.

Byte Stuffing



Note: *Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.*

Review of Byte stuffing



E.g. of byte stuffing when data to be transmitted is: WE WILL NOT ESCAPE FROM THE QUIZ FOR THE FLAG HOSTING

Data to be sent

WE WILL NOT ESCAPE FROM THE QUIZ FOR THE FLAG HOISTING

Frame sent

FLAG	HEADER	WE WILL NOT **ESCAPE** ESCAPE FROM THE QUIZ FOR THE **ESCAPE** FLAG HOISTING	TRAILER	FLAG

Frame received

FLAG	HEADER	WE WILL NOT **ESCAPE** ESCAPE FROM THE QUIZ FOR THE **ESCAPE** FLAG HOISTING	TRAILER	FLAG

Data after unstuffing

WE WILL NOT ESCAPE FROM THE QUIZ FOR THE FLAG HOISTING

- **Is ASCII code sufficient for all the characters used today in Internet?**

- **Is ASCII code sufficient for all the characters used today in Internet?**

Ans: ***No, because ASCII code is of 7-bits, and using these 7-bits, we can represents only 128 characters.***

So, most of the time, we use UNICODE which is of 16-bits and 32-bits.

So, we need to move to other stuffing techniques……….

Variable-Size Framing: Bit-Oriented Protocol

- In a bit-oriented protocol, the data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, etc.
- Apart from header and trailer, we still need a delimiter, called flag to separate one frame from other.
- Most of the protocols use a special 8-bit pattern flag **01111110** as the delimiter to define beginning and end of the frame.

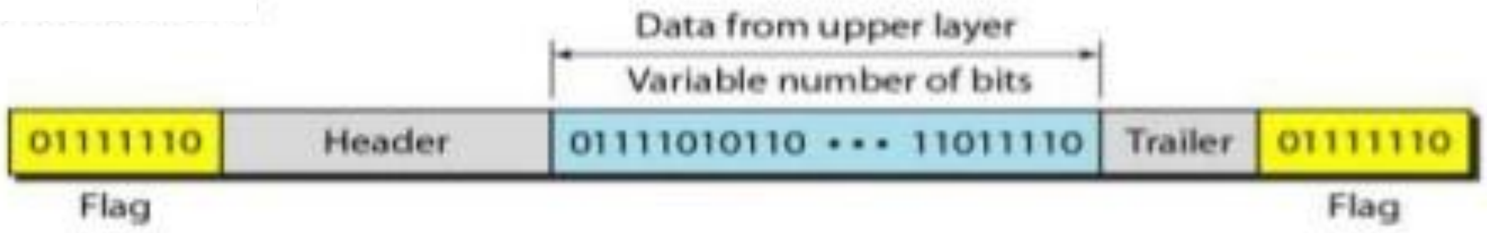
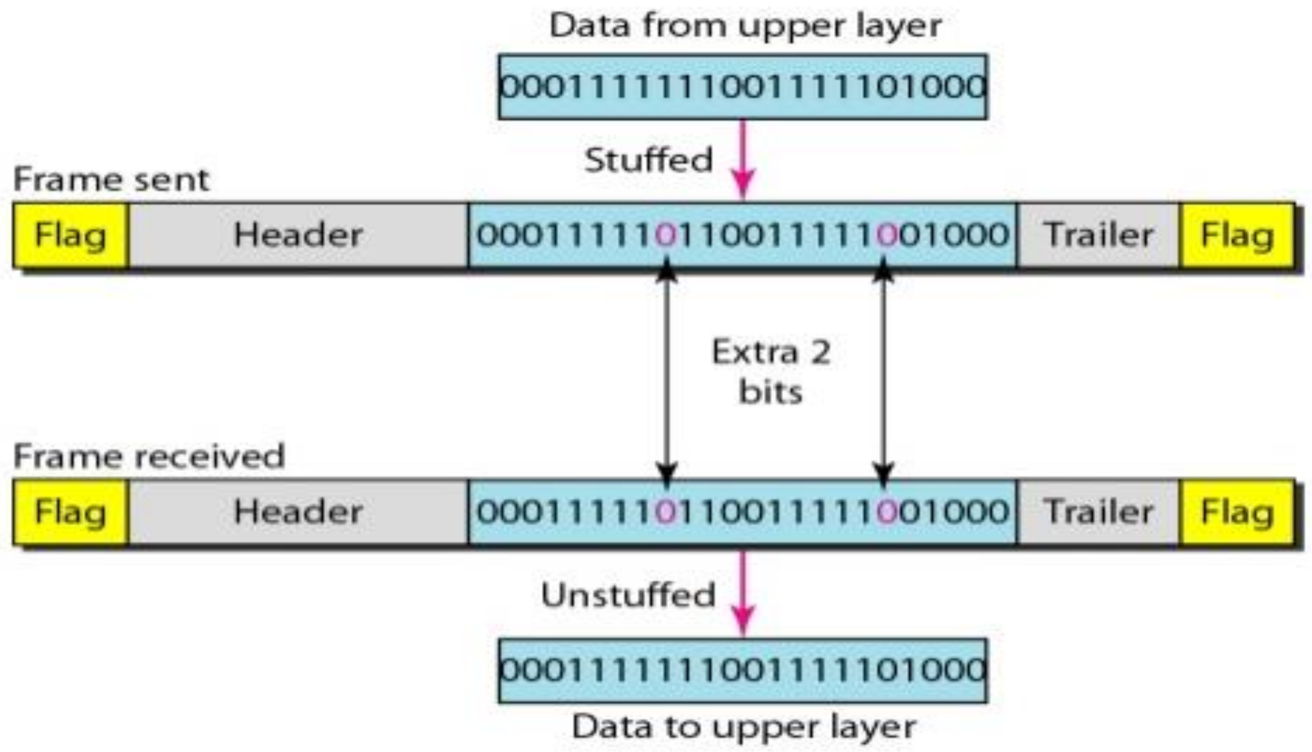


Fig. A frame in bit-oriented protocol

- **A Challenge:** The same type of problem (as happened in byte-oriented protocol) can occur here, if the flag pattern occurs in the data.
- **Sol:** *This problem is solved here by stuffing a single bit (instead of one byte in character oriented protocol) to prevent the pattern looking like a flag. This process is called bit-stuffing.*
- **Bit stuffing:** In this process, one extra 0 is added whenever five consecutive 1s follow a 0 in the data. This extra stuffed bit is eventually removed from the data by receiver.

Bit stuffing and unstuffing



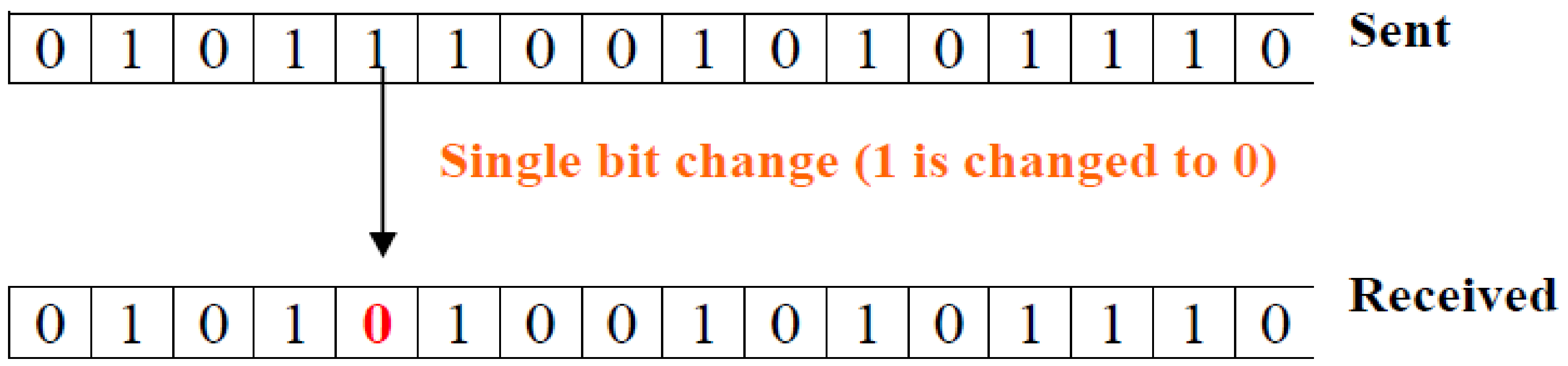
Note: A bit-oriented protocol is actually implemented by using the High-level Data Link Control (**HDLC**) Protocol.
Whereas a popular byte-oriented protocol is implemented by using Point-to-Point Protocol (**PPP**).

Error Control→ Error Detection

- Error control requires because
  - there are some applications which can tolerate certain level of error like video and voice applications.
  - But, there are some applications which can not tolerate error of any level like data applications.
- In data communication, there are **two types of errors**:
  1) *Single-Bit Error*
  2) *Burst Error*
- Error control has two phases:
  1) Error detection &
  2) Error correction

Types of Error: Single-Bit Error

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

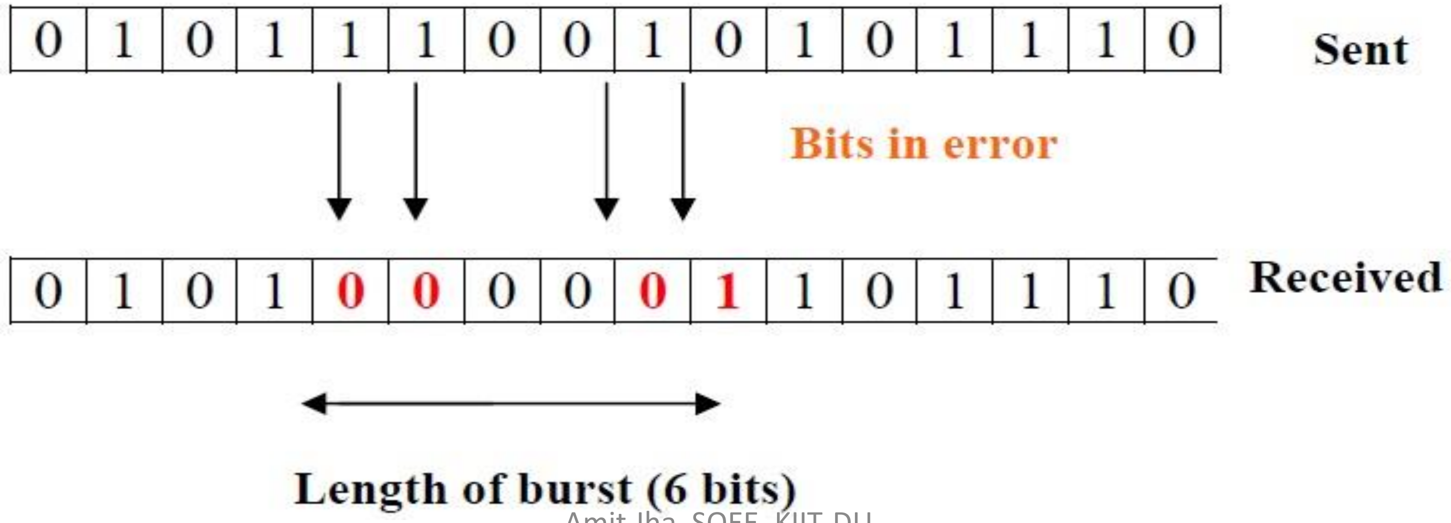


- Single-bit errors are the least likely type of error in serial data transmission.
- To understand why, imagine data sent at 1 Mbps. This means that each bit lasts only 1/1,000,000 s, or 1 µsec.
- For a single-bit error to occur, the noise must have a duration of only 1 µsec, which is very rare; noise normally lasts much longer than this.

- However, a single-bit error can happen if we are having a parallel data transmission.
- For example, if 16 wires are used to send all 16 bits of a word at the same time and one of the wires is noisy, one bit is corrupted in each word.

Types of Error: Burst Error

- 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.
- Note that burst error doesn't necessary means that error occurs in consecutive bits.
- The length of the burst error is measured from the first corrupted bit to the last corrupted bit. Some bits in between may not be corrupted.



- A burst error is more likely to occur than a single-bit error.
- This is because, the duration of noise is generally more 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 e.g., if we are sending data at 1kbps, a noise of 1/100 sec duration can affect 10 bits.

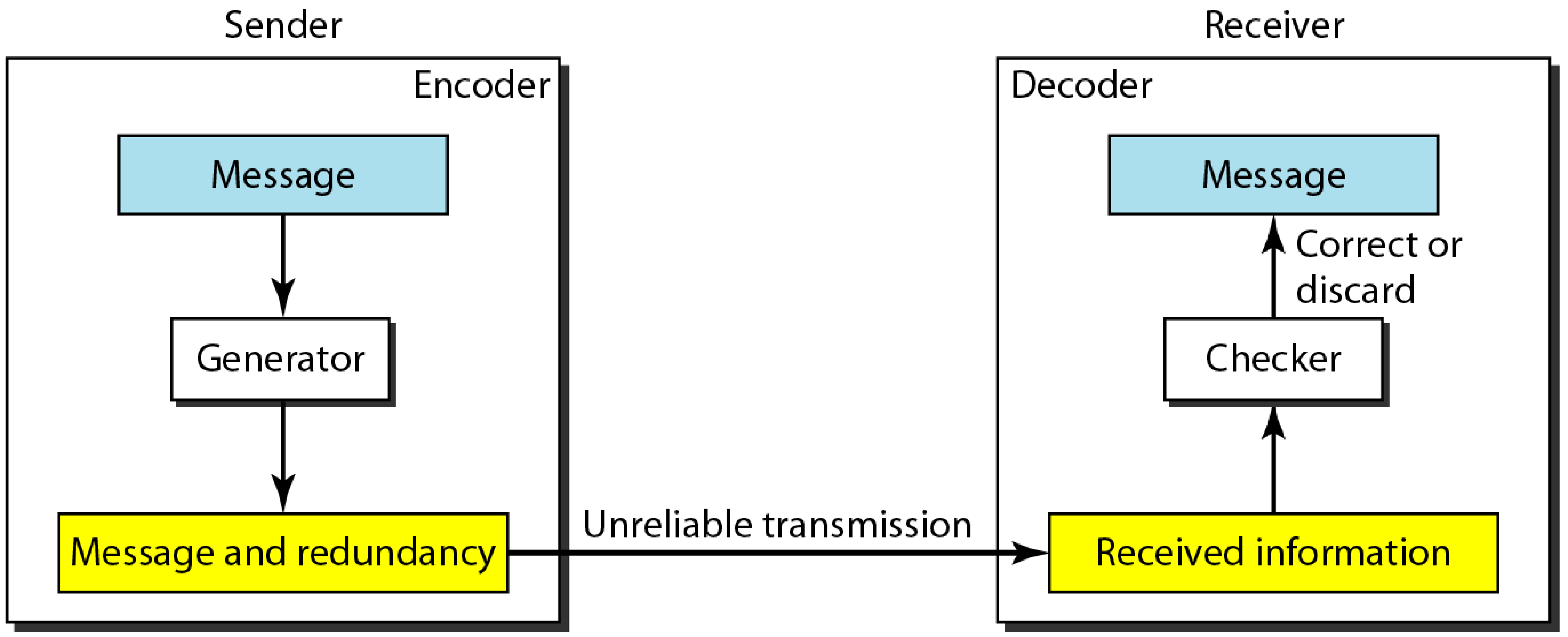
Let, A be **error detection** and B be **error correction**, then which of the following holds true?

a) A is easier than B b) A is more difficult than B

c) A is easier than B for less than 10 bits of data

d) A is more difficult than B for less than 10 bits of data

The structure of encoder and decoder



Introduction to Block Code

- *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.*

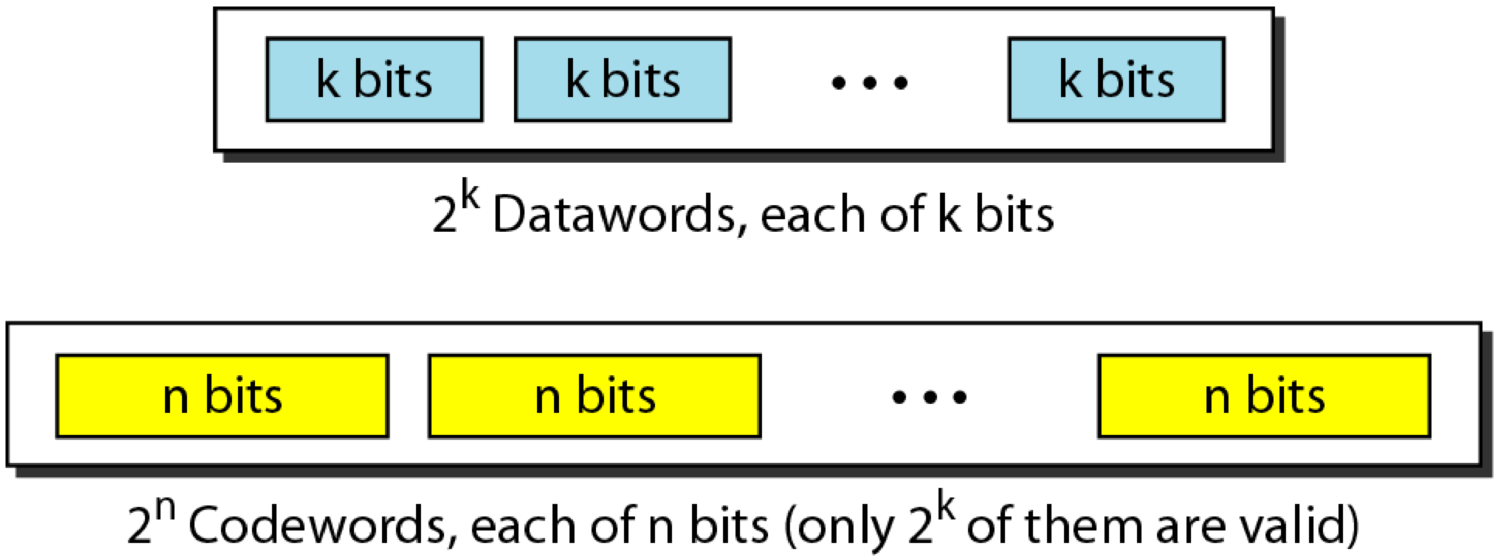

k bits
k bits
• • •
k bits
2^k Datawords, each of k bits
n bits
n bits
• • •
n bits
2^n Codewords, each of n bits (only 2^k of them are valid)

Fig: Datawords and codewords in block coding

Block Coding: Key points

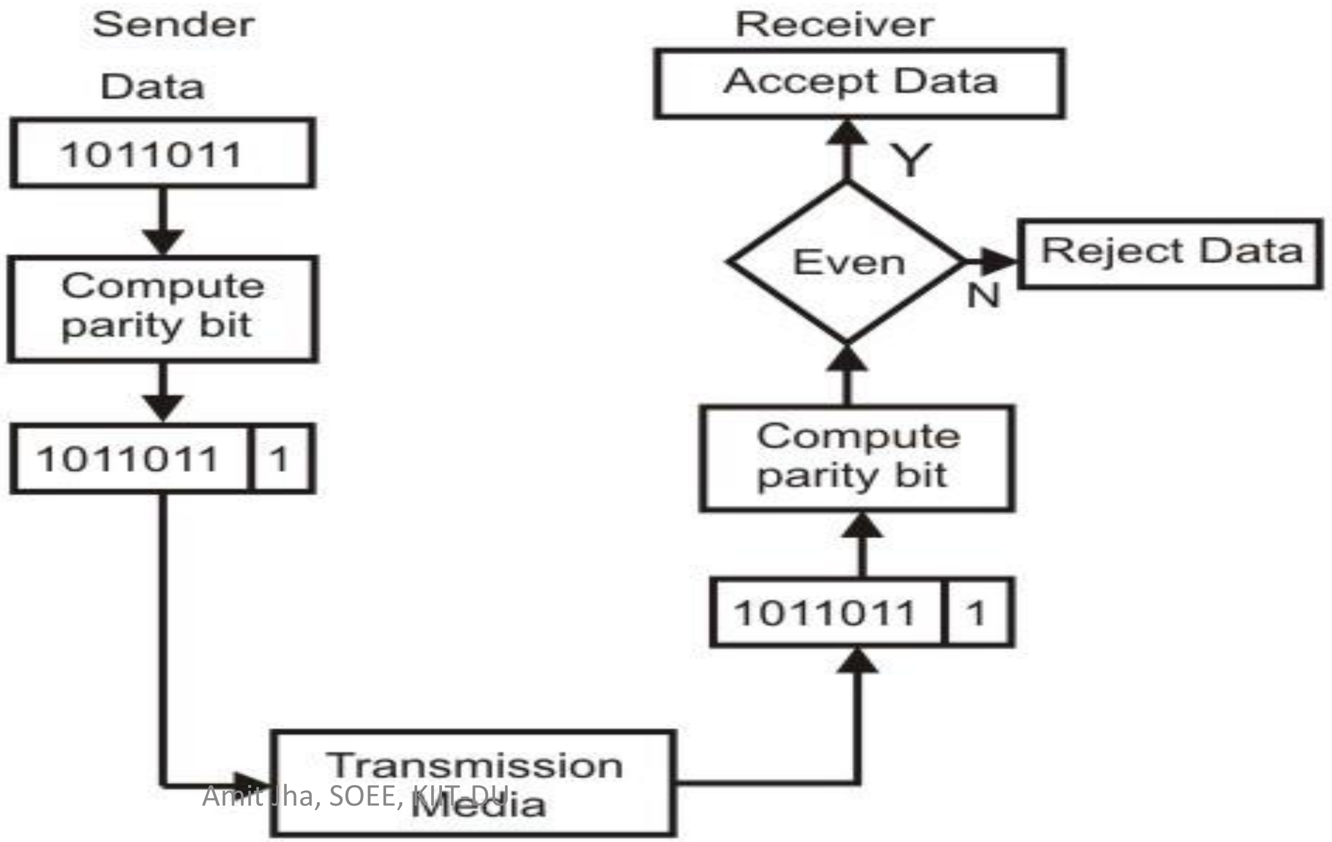
- *In 4B/5B coding scheme,*
  $k = 4$ and $n = 5$.
- *As we saw, we have $2^k = 16$ datawords and $2^n = 32$ codewords. We saw that 16 out of 32 codewords are used for message transfer and the rest are either used for other purposes or unused.*

Error Detecting Codes

- The central concept in detecting or correcting errors is ***redundancy,*** which means adding some extra bits along with data**.**
- The sender adds some redundant bits, whereas receiver removes it.
- Redundant bits (extra bits) facilitate detection and corrections of errors.
- Popular techniques for error detection are:
    1. Simple parity check
    2. Two-dimensional parity check
    3. Checksum
    4. Cyclic redundancy check

Error Detecting Codes: Simple parity check

- It is also known as **one-dimensional parity check**.
- In this technique, a redundant bit called **parity bit,** is appended to every data unit so that number of 1's in the unit (including the parity bit) becomes even.



Even parity checking scheme

- *At the Transmitter side*, parity bit generator adds
  - 1 if the data block contains odd number of 1’s
  - 0 if the data block contains even number of 1’s
- *At the receiver side*, parity bit is computed from the received data bits.
- If the receiver finds odd number of 1’s in received data, then it ensures that an error has occurred.

- For an e.g., the complete list of data words (for 4-bit) and the corresponding code words are given below:

Decimal value	Data Block	Parity bit	Code word
0	0000	0	00000
1	0001	1	00011
2	0010	1	00101
3	0011	0	00110
4	0100	1	01001
5	0101	0	01010
6	0110	0	01100
7	0111	1	01111
8	1000	1	10001
9	1001	0	10010
10	1010	0	10100
11	1011	1	10111
12	1100	0	11000
13	1101	1	11011
14	1110	1	11101
15	1111	0	11110

Key points

- It is also possible to use ***odd-parity*** checking, where the number of 1’s should be odd.
- From the last table it can be observed that
  - If we move from one code word to another, at least 2 data bits should be changed.
  - Thus, these set of code words are said to have a minimum distance ***(hamming distance)*** of 2.
  - So, if there exists one bit error then receiver will be able to detect it, but it can’t detect two bit error.
  - This is because, a code word with **two bits error** again becomes a valid member of the set.
- For a linear code, $\boxed{\textit{number of errors detected} = d_{min} - 1}$

where, dmin= minimum hamming distance

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

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

$000 \oplus 011$ is 011 (two 1s)

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

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

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

Q) Find the minimum hamming distance for the following codewords.

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

- **Sol:**

We first find all Hamming distances.

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

The d_{min} in this case is 2.

Q) Find minimum hamming distance for the codewords given in the following table.

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

Q) Find minimum hamming distance for the codewords given in the following table.

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

Ans: 3

- ***Will this method suffice our requirements of detecting errors?***

- ***Will this method suffice our requirements of detecting errors?***

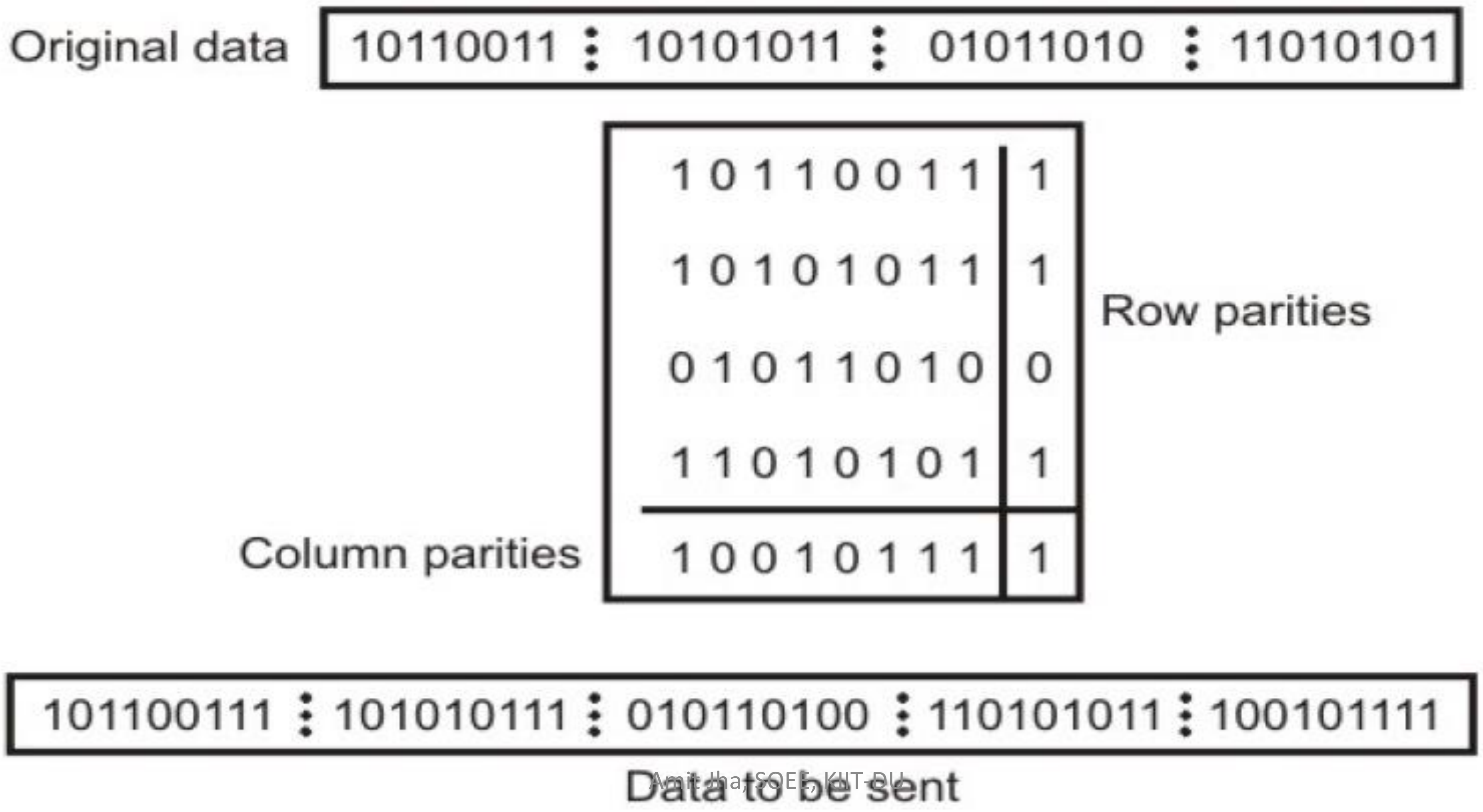
Ans: *No, because it is able to detect only a single-bit error.*

Explanation: *Let us assume that you are sending **01100**, but because of noise, it has been received as **10100**. In this scenario, receiver will not be able to identify the received data is in error because 10100 is a valid member of its code word set.*

Error Detecting Codes: Two-dimensional parity check

- In a two-dimensional parity check, the block of bits are organized in the form of a table.
- Parity check bits are calculated
  - For each row
  - as well as for each column
- Then, the complete table is sent in the form of block including data as well as both parity bits.
- At the receiver side, these are calculated with the parity bits calculated on the received data.
- The existence of odd number of 1’s ensures that the data are corrupted.

Two-dimensional parity checking



Key points

- Two-dimension parity checking increases the likelihood of detecting burst error.
- A 2-D parity check of n bits can detect a burst error of n bits.

- ***Will this method suffice our requirements of detecting errors?***

• *Will this method suffice our requirements of detecting errors?*

Ans: *No….*

Because if two bits in one data unit is damaged and two bits in another data unit is damaged at exactly same position, the 2-D parity checker will not detect any error.

Explanation: *Let us assume that data to be sent after parity is*
101100111 : 101010111 : 0101101100 : 110101011 : 100101111

But due to noise we receive the following data:

101100111 : 001110111 : 0101101100 : 110101011 : 000001111

Error Detecting Codes: Checksum

- In checksum error detection scheme, the data is divided into k segments each of m bits.
- Traditionally, the Internet has been using a 16-bit checksum, called ***Internet checksum***.

Concept behind *checksum*

- Its very simple……….
- Suppose our data 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 sums are same then receiver assumes no error and accepts the data; otherwise discard it.
- Now, we can further reduce the burden of receiver as follows:
  - Instead of sending sum, we send negative of sum, called ***checksum.*** In this case, we send (7, 11, 12, 0, 6, -36). The receiver adds all the numbers received (including the checksum). If result is 0, it assumes no error, otherwise there is an error.

How to calculate the Internet Checksum

- The sender calculates the checksum as follows:
  - The message is divided into 16-bit words.
  - The value of checksum word is set to 0.
  - All words including the checksum are added using one’s complement addition.
  - The sum is complemented and becomes the checksum.
- The checksum segment is sent along with the data segments.
- The receiver uses the following steps to detect error:
  - The message (including checksum) is divided into 16-bit words.
  - All words are added using **one’s complement addition**.
  - The sum is complemented and becomes the new checksum.
  - If the value of checksum is 0, the message is accepted; otherwise it is rejected.

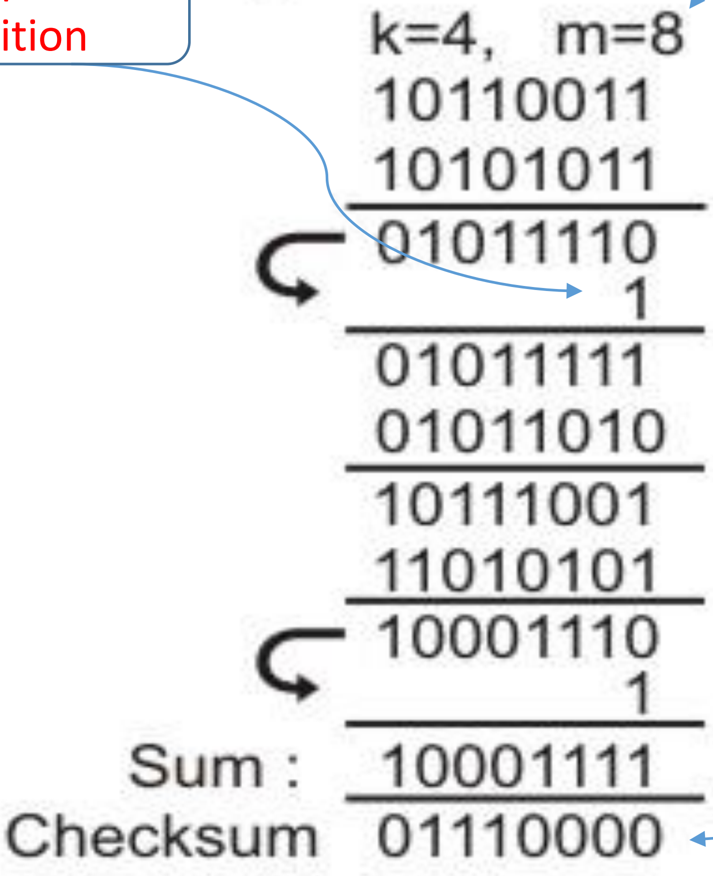
Ex.1: *For the given data,***10110011101010110101101011010101,** *calculate the checksum by dividing the data into words of 8 bits. Also, find whether the error has occurred or not, if the received data is*

a) **10110011101010110101101011010101**
b) **10110011101010110101101011010011**

- **Sol:** for part a)

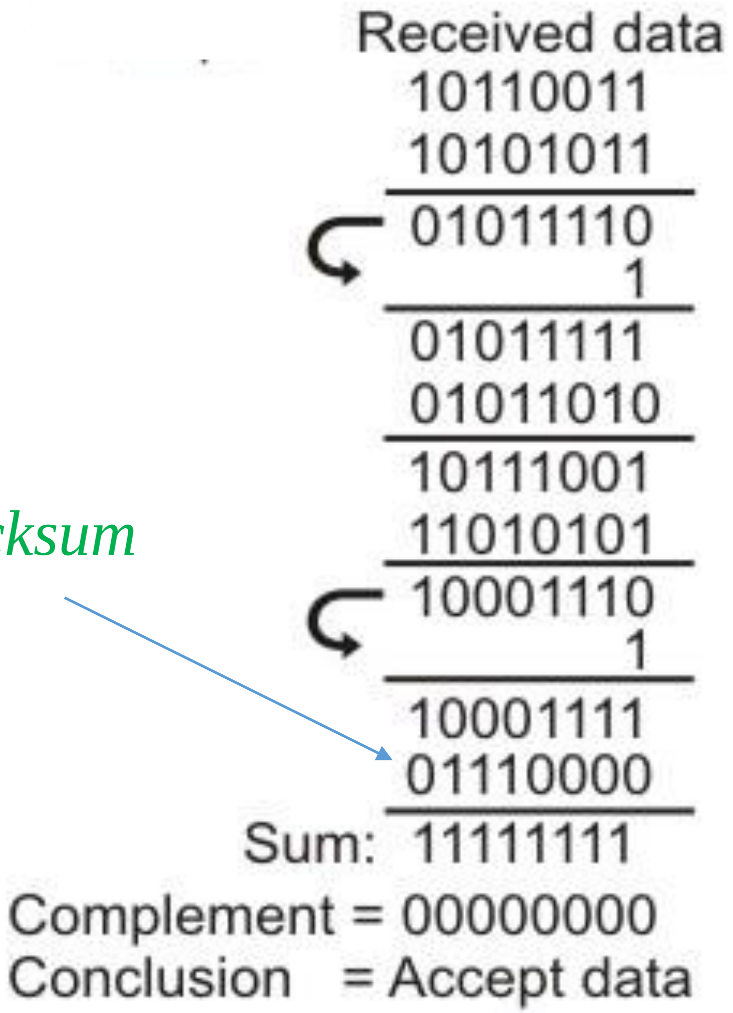
Since, 1's complement addition

Word size= 8bits



```
         k=4,   m=8
          10110011
          10101011
          --------
     ↺    01011110
                 1
          --------
          01011111
          01011010
          --------
          10111001
          11010101
          --------
     ↺    10001110
                 1
          --------
Sum :     10001111
          --------
Checksum  01110000
```

Old checksum



```
            Received data
              10110011
              10101011
              --------
          ↺   01011110
                     1
              --------
              01011111
              01011010
              --------
              10111001
              11010101
              --------
          ↺   10001110
                     1
              --------
              10001111
              01110000
              --------
         Sum: 11111111
Complement = 00000000
Conclusion   = Accept data
```

- **Solution of part b)**

Please try by your own

- **Ex 2:** *Calculate the checksum for a text of 8 characters ("Forouzan"). Assume that the checksum is of 16-bit, as used in Internet today. The ASCII code (in Hex) for the character used here is:* F=0x46, o=0x6F, r=0x72, u= 0x75, z= 0x7A, a=0x61, n=0x6E.

- **Sol:**

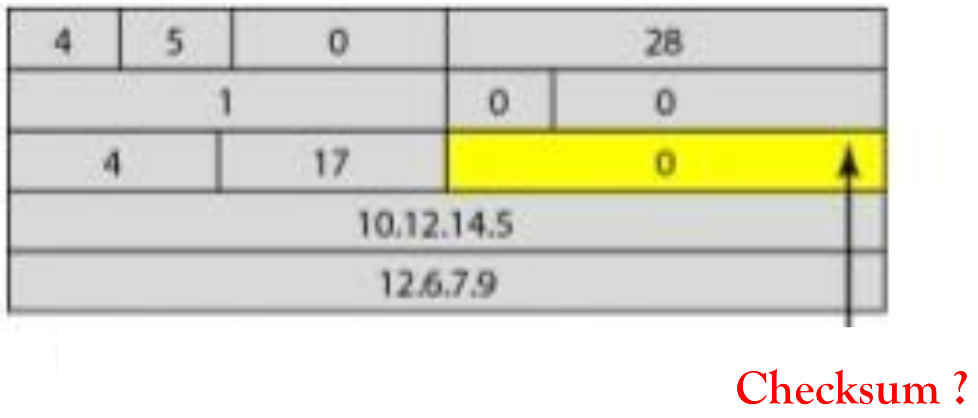
1	0	1	3	Carries
4	6	6	F	(Fo)
7	2	6	F	(ro)
7	5	7	A	(uz)
6	1	6	E	(an)
0	0	0	0	Checksum (initial)
8	F	C	6	Sum (partial)
			1	
8	F	C	7	Sum
7	0	3	8	Checksum (to send)

a. Checksum at the sender site

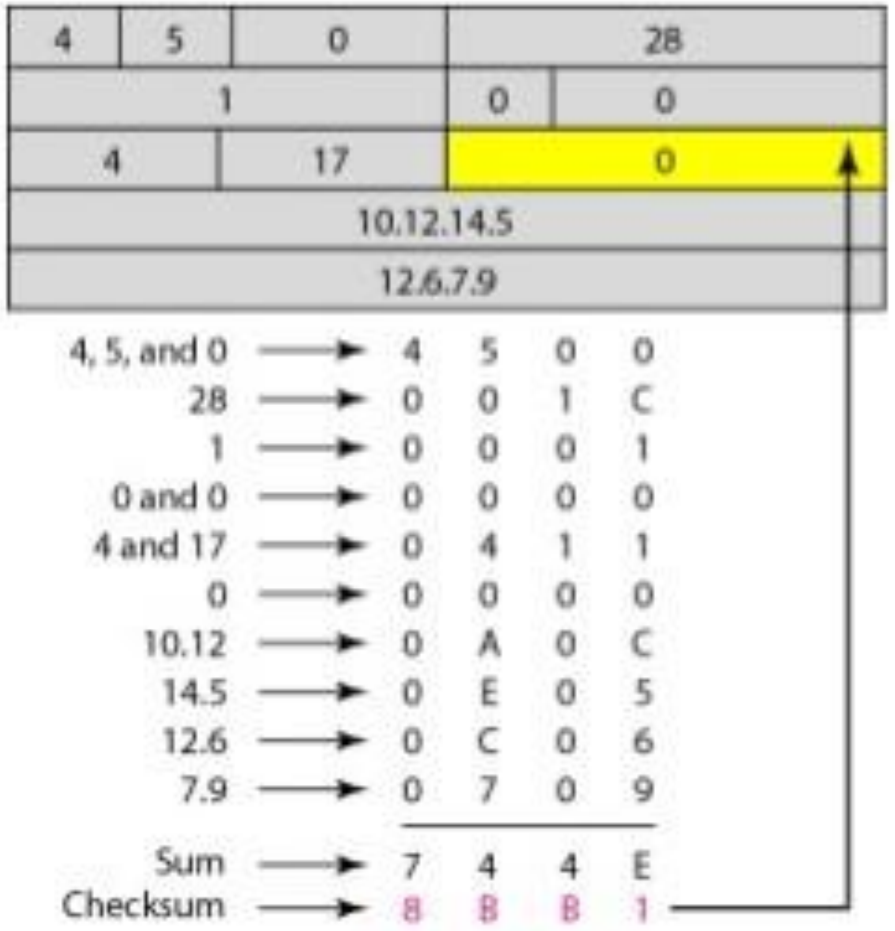
1	0	1	3	Carries
4	6	6	F	(Fo)
7	2	6	7	(ro)
7	5	7	A	(uz)
6	1	6	E	(an)
7	0	3	8	Checksum (received)
F	F	F	E	Sum (partial)
			1	
F	F	F	F	Sum
0	0	0	0	Checksum (new)

a. Checksum at the receiver site

Example 5: Calculate the checksum for IPv4 header shown below:



Problem: Calculate the checksum for IPv4 header shown below:



4, 5, and 0	4	5	0	0
28	0	0	1	C
1	0	0	0	1
0 and 0	0	0	0	0
4 and 17	0	4	1	1
0	0	0	0	0
10.12	0	A	0	C
14.5	0	E	0	5
12.6	0	C	0	6
7.9	0	7	0	9
Sum	7	4	4	E
Checksum	8	B	B	1

Ex 2: *Calculate the checksum for a text of 8 characters "I LIKE DCN". Assume that the checksum is of 16-bit, as used in Internet today. The ASCII code (in Hex) for the character used here is:* I=0x49, L=0x4C, K=0x4B, E=0x45, D=0x44, C=0x43, N=0x4E, space= 0x20, H=0x48, T=0x54, A=0x41.

Also, show that how will you verify that errors have been occurred if you receive the message as "I HATE DCN".

Please try by your own…………….

Internet checksum: Key points

- It is used in several Internet protocols like **IP, TCP, UDP** to detect errors in IP header, or in the header and data for TCP/UDP.
- Generally, a checksum is calculated for header content and included in a special field.
- The checksum is calculated **at every router**.
- Overhead is less as compared to two-dimension parity check.

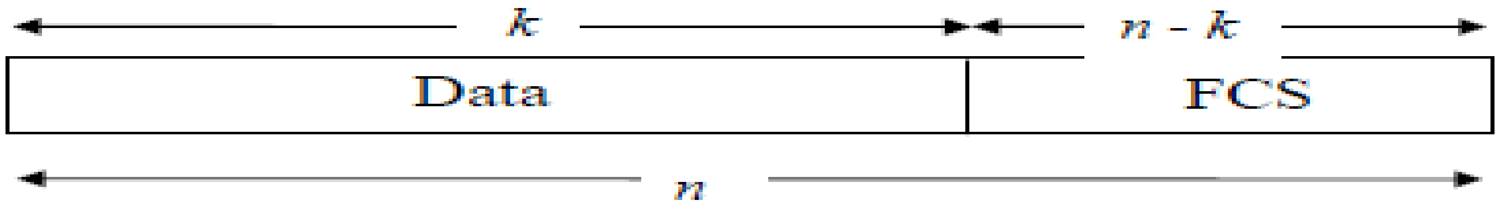
• **Disadvantage:** It will not be able to detect all errors if the total sum remains constant.

Error Detecting Codes: Cyclic Redundancy Check (CRC)

- One of the **most powerful** and commonly used error detecting codes.

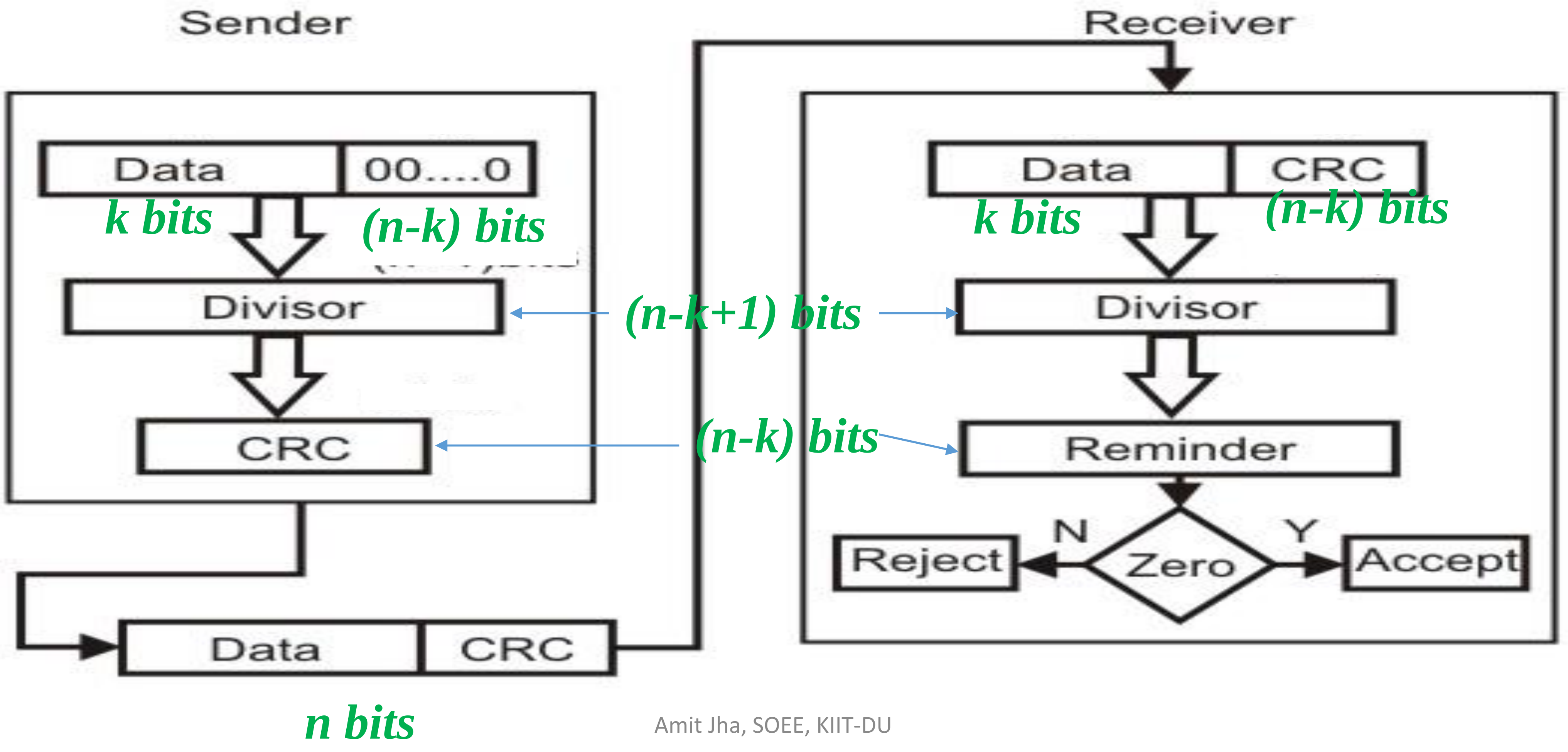
- Concept behind it………
  - Given a k-bit block of bit sequence, the sender generates an $(n-k)$ bit sequence, known as ***frame check sequence(FCS),*** so that the resulting frame, consisting of n bits, is exactly divisible by same ***predetermined number*** *(which is of $(n-k+1)$ bits)*.
  - The receiver divides the incoming frame by that number and, if there is no remainder, it assumes that there was no error.

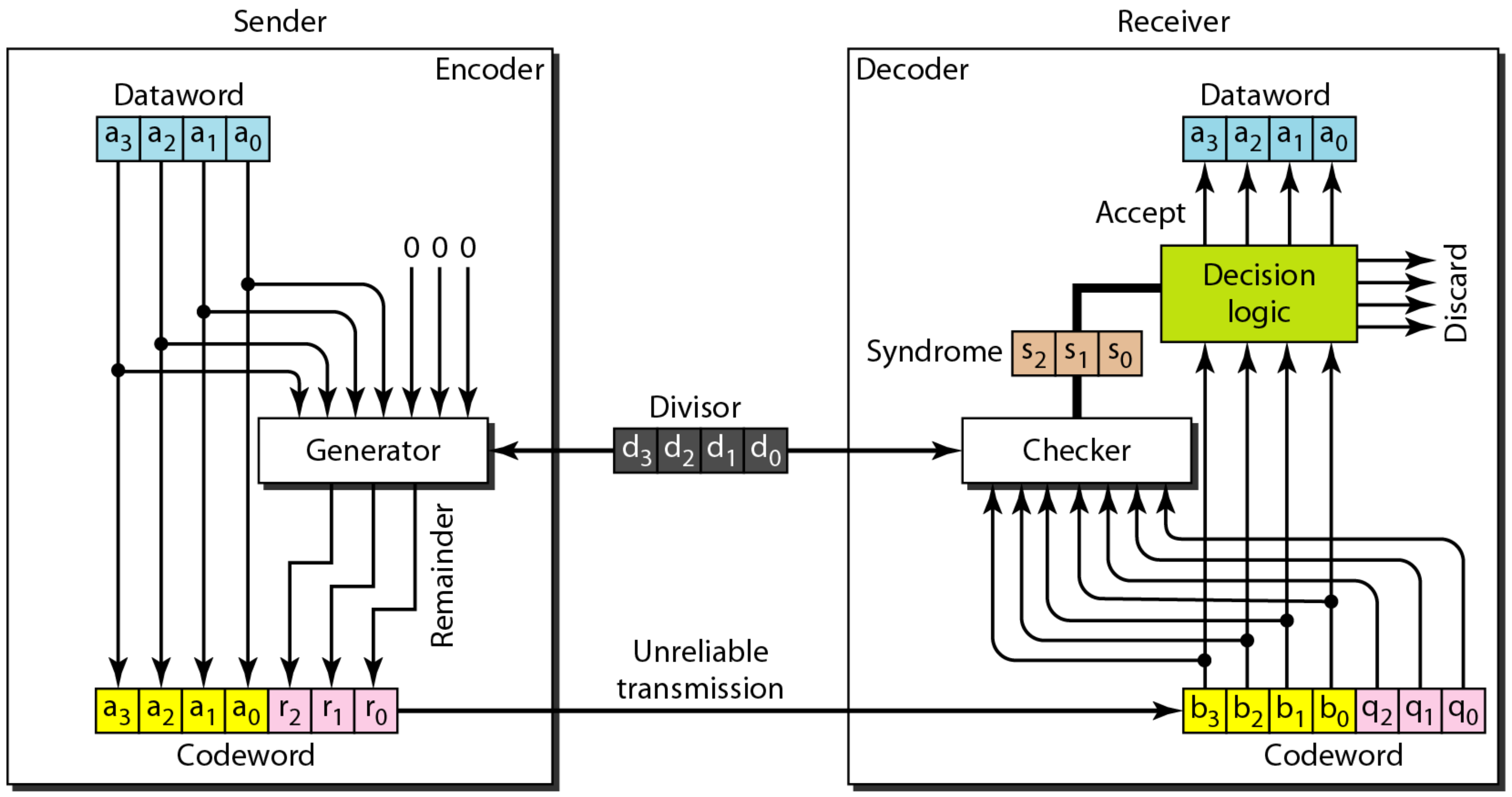
- Frame check sequence is also known as ***cyclic redundancy check bits.***



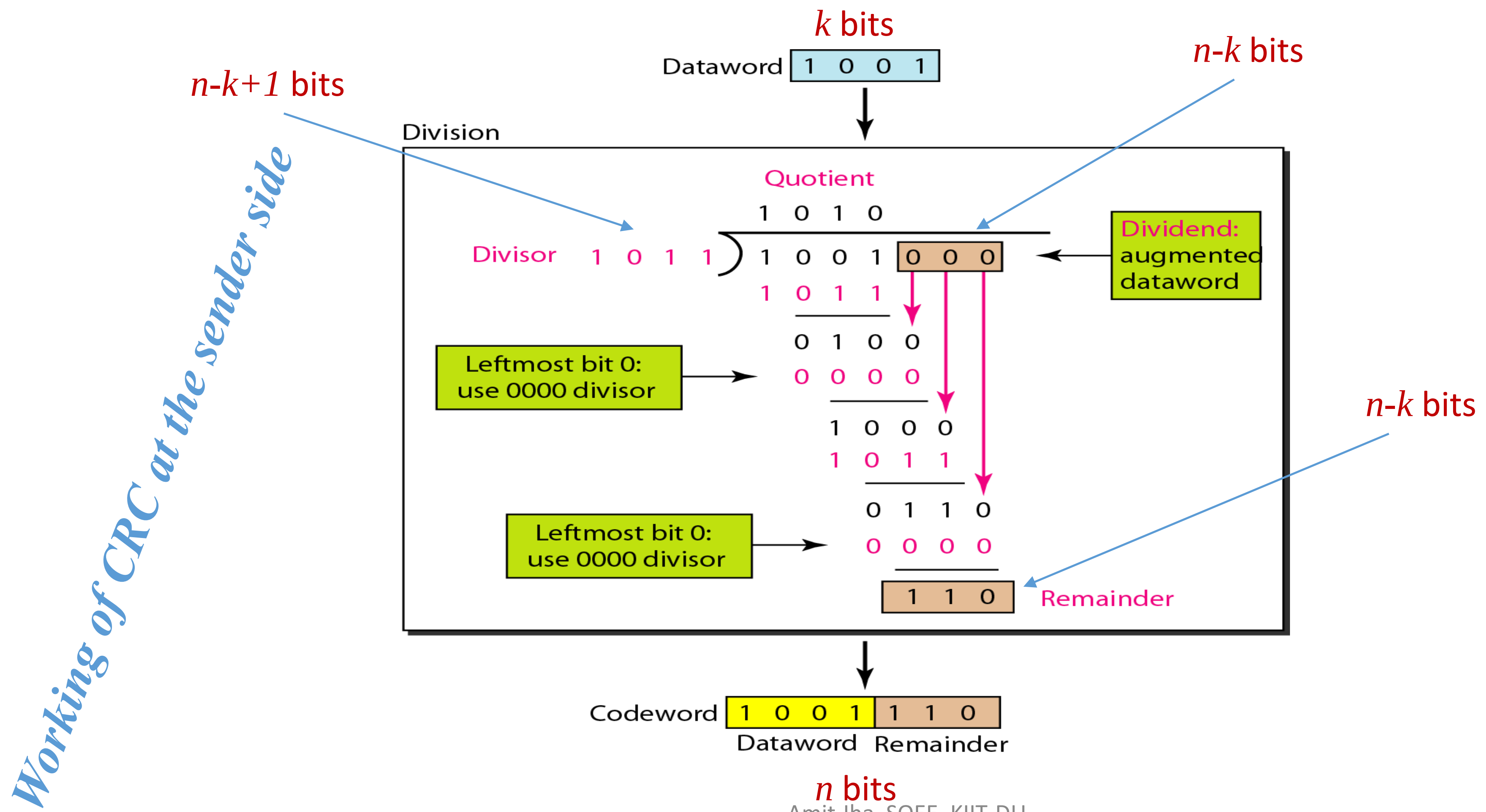
Frame

Basic scheme for Cyclic Redundancy Checking

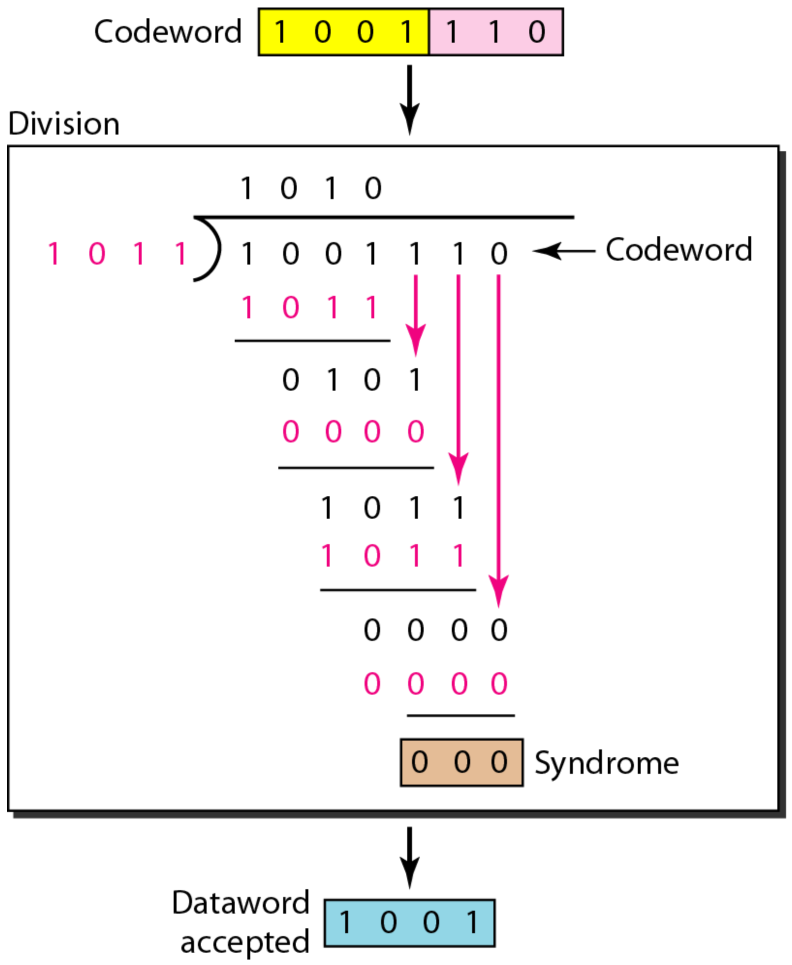



Sender
Receiver
Encoder
Decoder
Dataword
a_3 a_2 a_1 a_0
0 0 0
Generator
Remainder
Divisor
d_3 d_2 d_1 d_0
Checker
Syndrome
s_2 s_1 s_0
Decision logic
Accept
Discard
Unreliable transmission
a_3 a_2 a_1 a_0 r_2 r_1 r_0
Codeword
b_3 b_2 b_1 b_0 q_2 q_1 q_0
Codeword

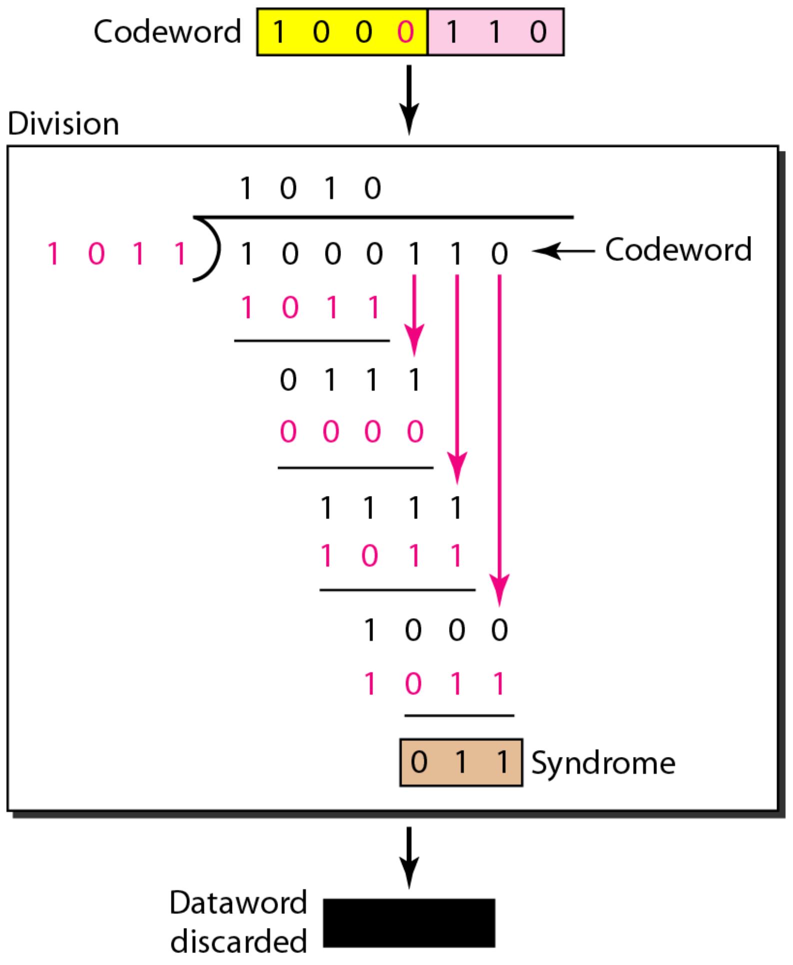
How to do a binary division !!!



Working of CRC at the receiver side



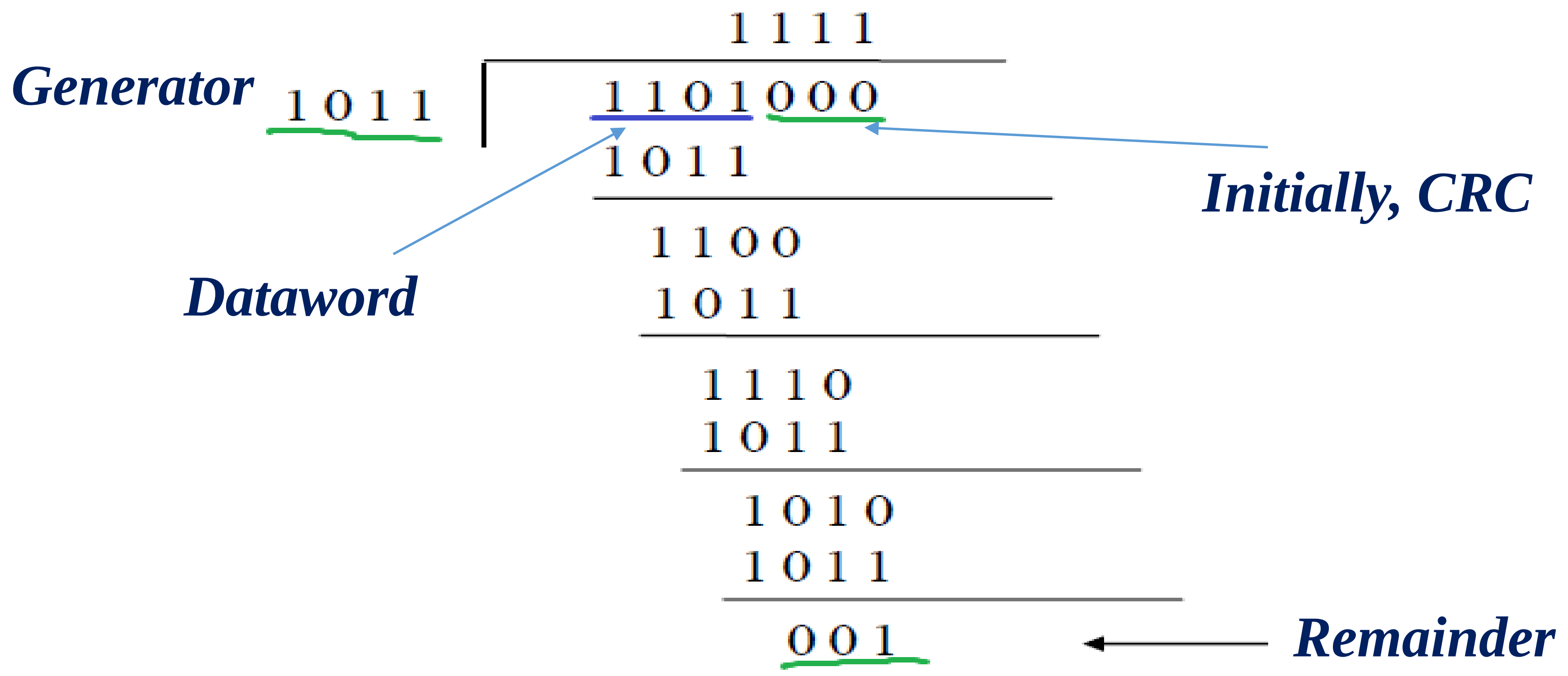
Case: 1



Case: 2

- **Ex:** Consider a case where data word is 1101 (i.e., m=4 bits) and predetermined number is 1011. Determine FCS.
- **Sol:**

- Initially, we choose CRC as all zero bits. Since, predetermined number is of 4 bits, we choose 3 zero bits, i.e., 000 as CRC.
- So, we append 3 zeros to dataword and then it will be divided by 1011.


1 1 1 1
Generator
1 0 1 1
1 1 0 1 0 0 0
1 0 1 1
Dataword
Initially, CRC
1 1 0 0
1 0 1 1
1 1 1 0
1 0 1 1
1 0 1 0
1 0 1 1
0 0 1
Remainder

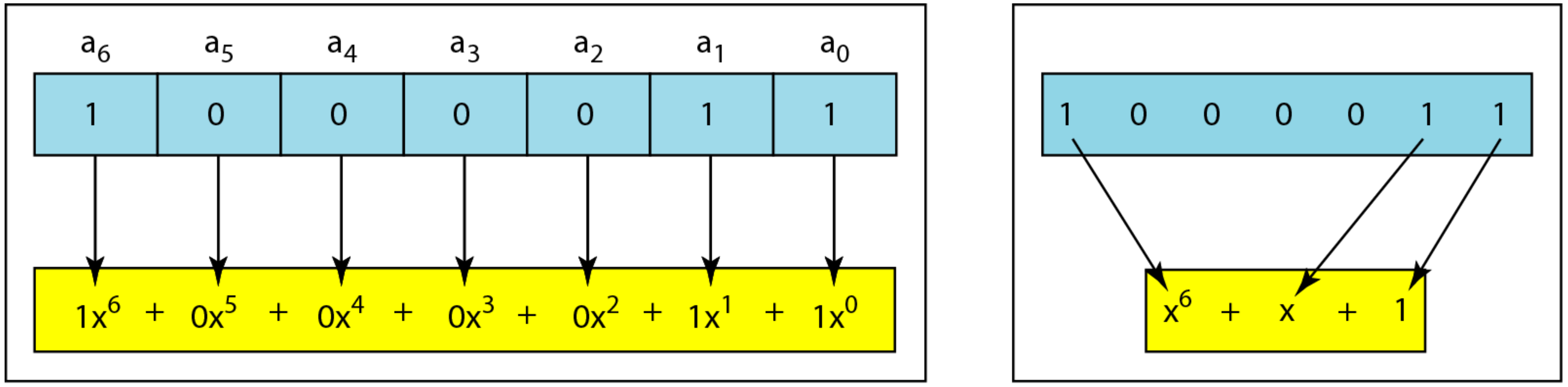
- Q) Find CRC/FCS for message **1010001101** for the given pattern, **110101**. Also verify whether the receiver will accept or reject, if the received data is **101000110001110.**

- **!!!!!!!!!!! Just do it !!!!!!!!**

CRC code using Polynomial

- It is a better way to understand cyclic code.
- Using polynomial, analysis of cyclic code is easier as compared to binary representation.
- We can use polynomial to represent a binary word.
- Each bit from right to left is mapped onto a power term.
- The rightmost bit represents the “0” power term. The bit next to it the “1” power term, so on.
- If the bit is of value zero, the power term is deleted from the expression.

A polynomial to represent a binary word



a_6 a_5 a_4 a_3 a_2 a_1 a_0
1 0 0 0 0 1 1
$1x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0$
1 0 0 0 0 1 1
$x^6 + x + 1$

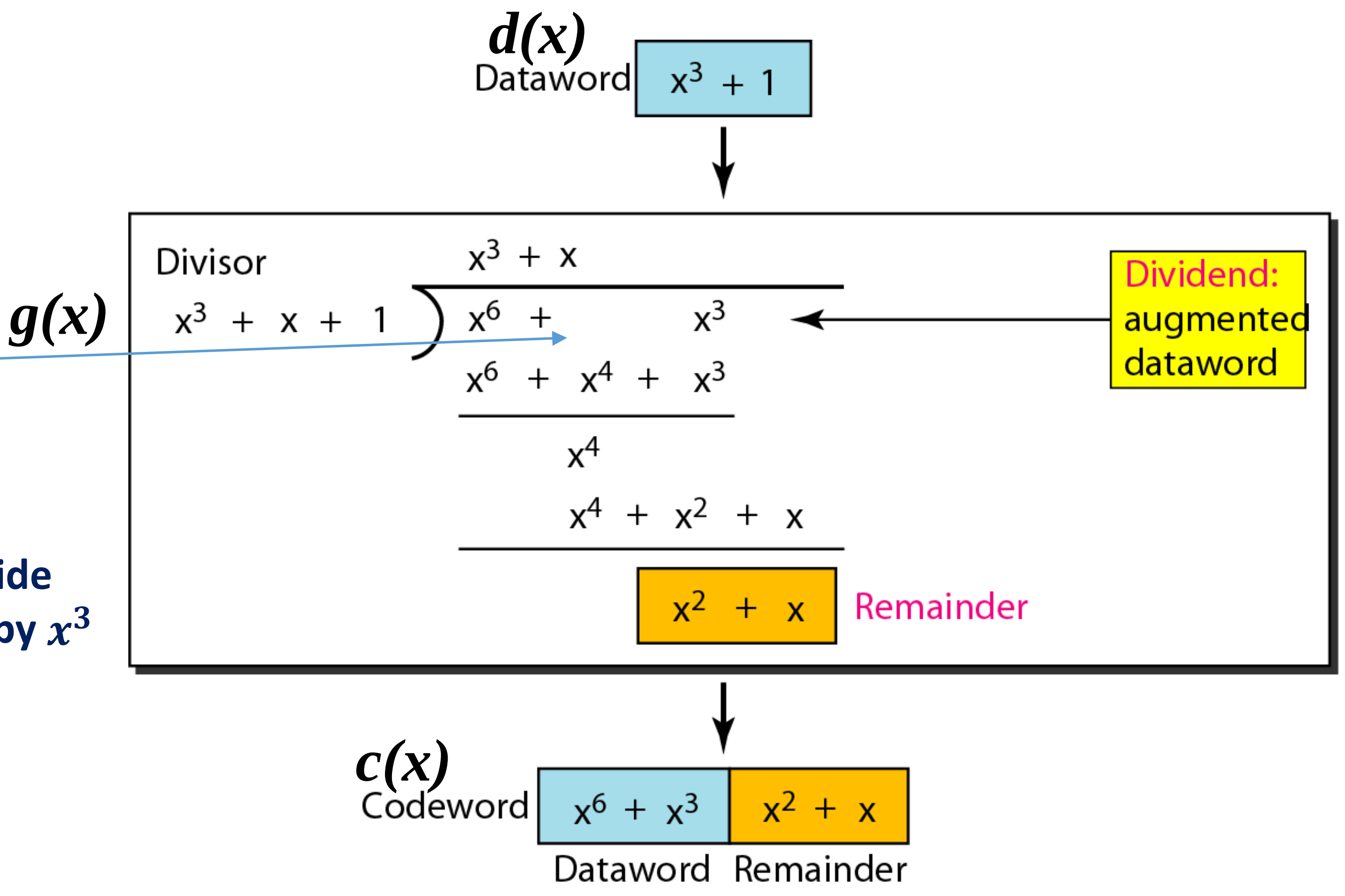
a. Binary pattern and polynomial b. Short form

Note: **for n bits, polynomial can have a maximum degree of n-1.**

- We use the following notations in polynomial representation.
  - *Dataword: d(x)*
  - *Codeword: c(x)*
  - *Error: e(x)*
  - *Generator: g(x)*
  - *Syndrome: s(x)*

- ***Syndrome:*** *It is the remainder produced by the checker (at Rx. side). It is also treated as set of patterns accepted. For e.g., generally in CRC, we say that if syndrome is all 0's then no error; otherwise there is some error.*

Ex: Consider 1001 be the dataword and 1011 be the pattern (generator). Find CRC using polynomial approach.



After adding 3 zeros

Adding 3 zeros to the right side is equivalent to multiplying by x^3

CRC code: Key points

- In a cyclic code,
  - Divisor is called generator polynomial or simply generator, denoted by *g(x)*.
  - Remainder at the receiver side is called syndrome, denoted by *s(x)*.
  - Dataword and remainder together are called codeword, denoted by *c(x)*.
  - Error polynomial is represented as *e(x)*.
  - Received codeword is ***c'(x)=c(x)+e(x).***
- In cyclic code, those e(x) errors that are divisible by g(x) are not caught.

Received codeword (c(x) + e(x))/g(x) = c(x)/g(x) + e(x)/g(x)

- Properties of the CRC code are determined by the choice of the generator polynomial. Choose g(x) such that as many error patterns e(x) as possible may be detected.

How to choose generator polynomial…

- Two main properties which must be satisfied by a generator polynomial.
    1) It should not be divisible by X.
    2) It should not be divisible by (X+1).

Performance:

- CRC can detect
    - All single-bit errors
    - All double-bit errors
    - All burst errors of less than the degree of the polynomial.
    - Most of the larger burst errors with a high probability.
    - For e.g., CRC-12 detects 99.97% of errors with a length 12 or more.

Standard Generator Polynomials

CRC = cyclic redundancy check

CCITT = Consultative Committee for International Telephony and Telegraphy

CRC-8:

$= x^8 + x^2 + x + 1$

ATM

CRC-16:

$= x^{16} + x^{15} + x^2 + 1$
$= (x + 1)(x^{15} + x + 1)$

Bisync

CCITT-16:

$= x^{16} + x^{12} + x^5 + 1$

HDLC, XMODEM

CCITT-32:

$= x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

IEEE 802

Error Control → Error Correcting Codes

Two basic approach for error correction

- **Forward error correction:** In this method, receiver can use an error-correcting code, which automatically corrects certain errors.
- E.g., Hamming Code, BCH Code, etc.

- **Backward error correction:** In this method, when an error is discovered; the receiver can have the sender retransmit the entire data unit. This is also referred as *Retransmission* mechanism.
- E.g., Go-Back-N ARQ, Selective Repeat ARQ, etc.
- ***HYU:*** *Explain* ***error correction*** *Vs* ***retransmission*** *concept.*

Q) How many errors can be corrected by using error correcting codes?

Q) How many errors can be corrected by using error correcting codes?

Ans: *In theory, it is possible to correct any number of errors atomically.*
Error-correcting codes are more sophisticated than error detecting codes and require more redundant bits. The number of bits required to correct multiple-bit or burst error is so high that in most of the cases it is inefficient to do so.

For this reason, most error correction is limited to one, two or at the most three-bit errors.

In Hamming code, for 1-bit error

- **Requirements for error detection:** A code is an error detecting codes if and only if the minimum distance (Hamming distance) between any two code word is two.
- **Requirement for error correction:** For a code to be error-correcting, the minimum distance between any two code words must be more than two.

The # errors can be detected = dmin-1

and

The # errors can be corrected = (dmin-1)/2

Single-bit error correction: Hamming Code

Hamming Code: It is a binary linear block code denoted as *(n,k,d)*, *where*, $n \le 2^r - 1$, *and n=k+r.*

k= # message bits, *r= # parity bits*

Q.1) How to decide number of parity bits for a given data/message?

It is decided by, $\boxed{k + r \le 2^r - 1}$ **1**

Some examples:

Ex.1: If k =1 then r =2 satisfies result 1.

Ex.2: If k =2 then r =3 satisfies result 1.

Ex.3: If k =3 then r =3 satisfies result 1.

- **Ex.4:** If $k = 4$ then $r = 3$ satisfies result 1.
- **Ex.5:** If $k = 5,6,7,8$ then $r = 4$ satisfies result 1.

Observation: *The number of parity bits required may be same for two different data words with different number of bits.*

Q.2) What are the positions of parity bits?

Ans: Position of the parity bits is given by relation $\mathbf{2^r}$,where $r = 0,1,2,3 \ldots .$

Q.3) What are the values of parity bits?

Ans: For first parity bit, it is obtained by doing XOR operation of all those data bits whose position in binary has 1st bit 1 from the right side.

Simillarly, for ith parity bit, it is obtained by doing XOR operation of all those data bits whose position in binary has ith bit 1 from the right side.

- **Q.4) How an error is corrected?**

Ans: For received data, again calculate value of each parity bit, now there can be two cases:

Case 1: If there is error in any one of the parity bits.

The parity bit which will be in error will differ from the old parity bit**.**

Case 2: If there is error in any one of the data bits.

Any two or more parity bits will differ from the old parity bits. In this case, the position of data bit in error is calculated by doing the sum of positions of respective parity bits.

- **Example:** Let us assume that data to be transmitted be **1011** (here k=4).

- Since k=4, r=3 satisfies equation 1. So, we have to add 3 parity bits.

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	**0**	**1**		**1**		
111	110	101	100	011	010	001

$$\because \mathbf{2^0 = 1, 2^1 = 2, 2^2 = 4}$$

- Parity bits will be at positions 1,2 and 4.

- **Example:** Let us assume that data to be transmitted be **1011** (here k=4).

- **Values of parity bits:**

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1		1		
111	110	101	100	011	010	001

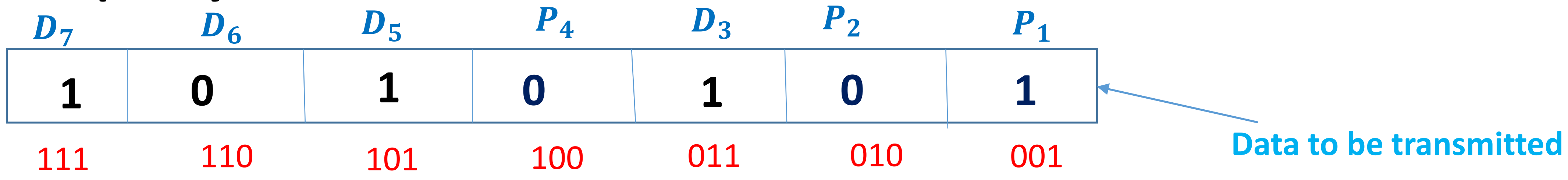
$$P_1 = D_3 \oplus D_5 \oplus D_7 = 1 \oplus 1 \oplus 1 = 1$$

$$P_2 = D_3 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$$

$$P_4 = D_5 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$$

- **Example:** Let us assume that data to be transmitted be **1011** (here k=4).

- **Values of parity bits:**



D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	0	1	0	1
111	110	101	100	011	010	001

$$P_1 = D_3 \oplus D_5 \oplus D_7 = 1 \oplus 1 \oplus 1 = 1$$

$$P_2 = D_3 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$$

$$P_4 = D_5 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$$

- **Example:** Let us assume that data to be transmitted be **1011** (here k=4).

- **Case 1: At the Rx side, if received data is**

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	1	1	0	1
111	110	101	100	011	010	001

$P_1 = D_3 \oplus D_5 \oplus D_7 = 1 \oplus 1 \oplus 1 = 1$ .... *matched*

$P_2 = D_3 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$ ....*matched*

$P_4 = D_5 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$ ....*not matched*

So, error is in the 4th position, which is a parity bit, P4

- **Case 2: At the Rx side, if received data is**

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	0	0	0	1
111	110	101	100	011	010	001

$P_1 = D_3 \oplus D_5 \oplus D_7 = 0 \oplus 1 \oplus 1 = 0$ ……………….***not matched***

$P_2 = D_3 \oplus D_6 \oplus D_7 = 0 \oplus 0 \oplus 1 = 1$ ……………….***not matched***

$P_4 = D_5 \oplus D_6 \oplus D_7 = 1 \oplus 0 \oplus 1 = 0$ ………………. ***matched***

So, position is 1+2=3, where 3 is position od data bit D3

Problem: *For 8 bits of data **11000010,** using hamming code, find*

a) The number of parity bits required.

b) The values of parity bits.

c) The data to be transmitted.

d) Correct the error, if received codeword is 110010010010.

!!!! Just do it !!!!

Quick review:

1. Error detection is usually done in _________ layer of OSI.
2. _________ uses the one’s complement arithmetic.
3. _______________ is the error detection method which consists of a parity bit for each data unit as well as an entire data unit of parity bit.
4. The number of bits position in which code words differ is called the ___________ distance.
5. To detect d errors, you need a distance _________ code.
6. To correct d errors, you need a distance ________ code.
7. ___________ error means that only one bit of given data unit (such as a byte, character, or data unit) is changed from 1 to 0 or from 0 to 1.

Quick review:

8. Which Error detection method can detect a burst error? ____________.

9. __________ involves polynomials.

10. In cyclic redundancy check, CRC is ___________.

11. In Cyclic Redundancy Check, the divisor is ______ the CRC.

12. When an error is discovered; the receiver can have the sender retransmit the entire data unit. This is known as ___________________ **correction**.

13. When receiver can use an error-correcting code, which automatically corrects certain errors. This is known as ______________________ **correction**.