IAT-3 Solutions

1.

Dataword	**Codeword**
00	**00000**
01	**01011**
10	**10111**
11	**11111**

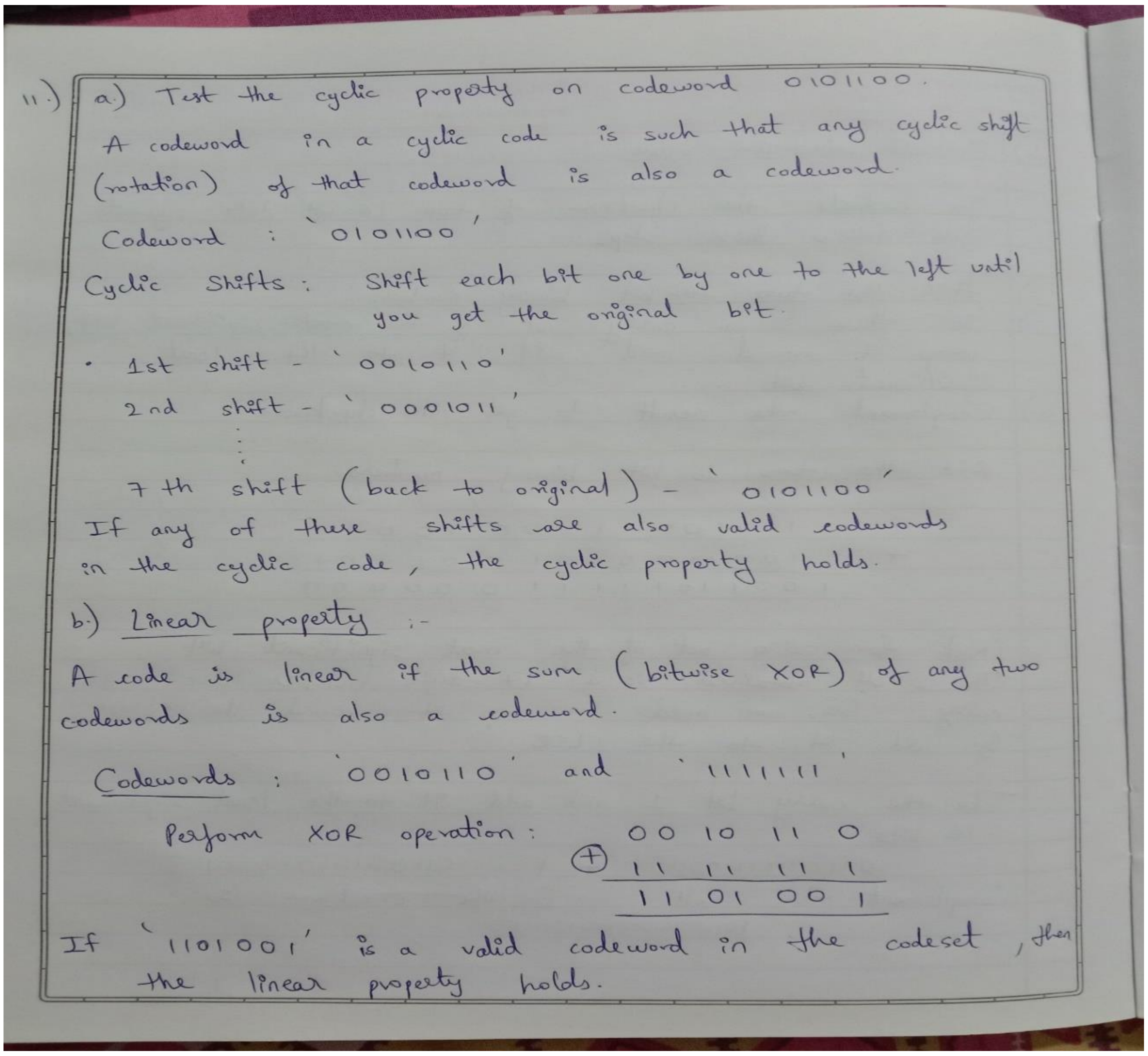
Explain Cyclic Redundancy Codes with diagram and Check the following properties-
a. Test the Cyclic Property on Codeword 0101100
b. Test the Linear property on codewords 0010110 and 1111111.

Answer- Cyclic Redundancy 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. 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:

$$b_1 = a_0 \quad b_2 = a_1 \quad b_3 = a_2 \quad b_4 = a_3 \quad b_5 = a_4 \quad b_6 = a_5 \quad b_0 = a_6$$

Cyclic Redundancy Check

We can create cyclic codes to correct errors.



ii.) a) Test the cyclic property on codeword 0101100.

A codeword in a cyclic code is such that any cyclic shift (rotation) of that codeword is also a codeword.

Codeword : '0101100'

Cyclic Shifts : Shift each bit one by one to the left until you get the original bit.

- 1st shift - '0010110'

2nd shift - '0001011'

⋮

7 th shift (back to original) - '0101100'

If any of these shifts are also valid codewords in the cyclic code, the cyclic property holds.

b.) Linear property :-

A code is linear if the sum (bitwise XOR) of any two codewords is also a codeword.

Codewords : '0010110' and '1111111'

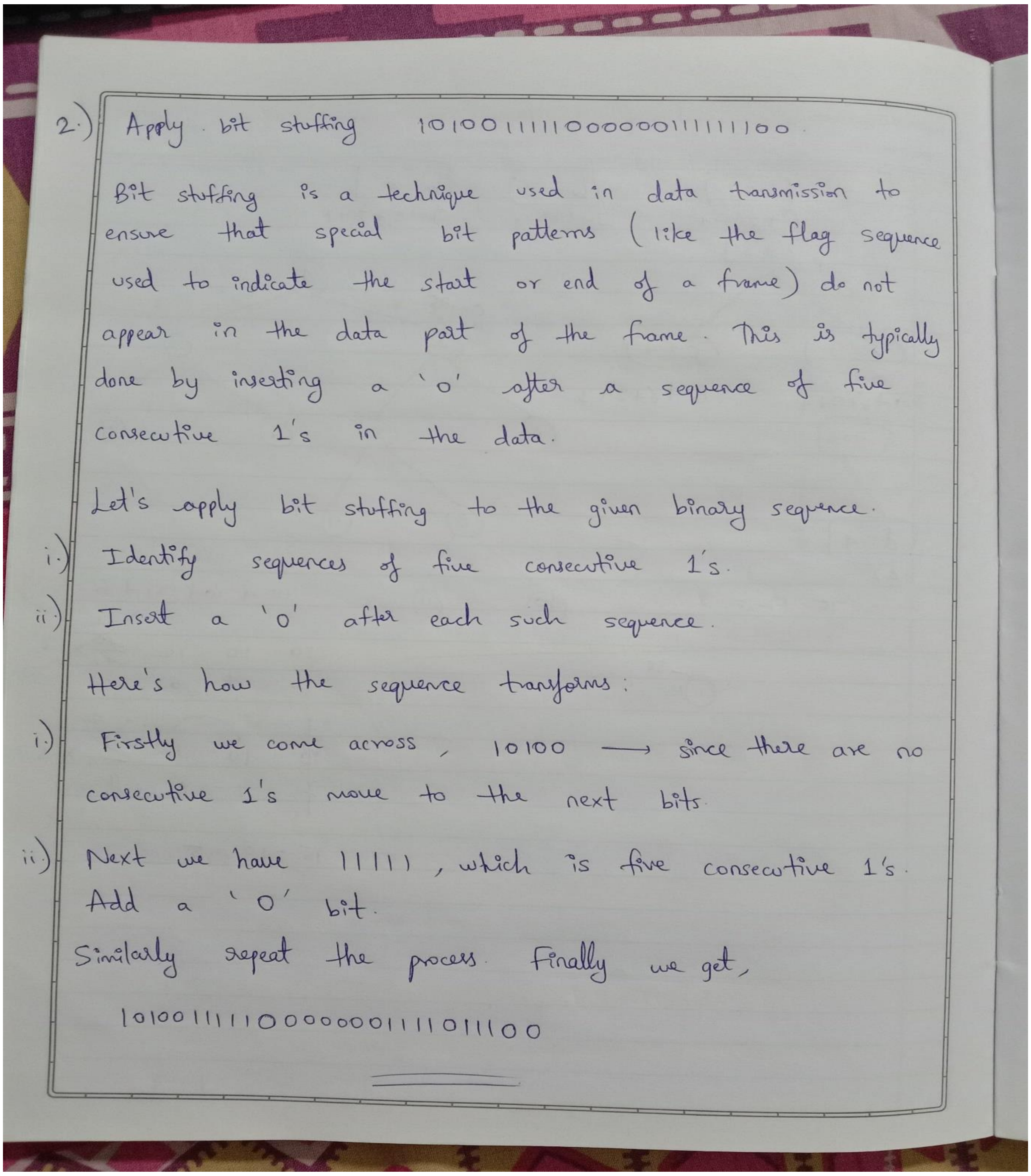
Perform XOR operation :

$$\begin{array}{r} 00\ 10\ 11\ 0 \\ \oplus\ 11\ 11\ 11\ 1 \\ \hline 11\ 01\ 00\ 1 \\ \hline \end{array}$$

If '1101001' is a valid codeword in the codeset, then the linear property holds.

2. Differentiate between fixed and variable size framing. Explain Character Stuffing and Bit Stuffing process. Apply bit stuffing for the following data bit Sequence. 1010011111000000111111100.

Fixed Size Framing- Frames can be of fixed or variable size. 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- Our main discussion in this chapter concerns variable-size framing, prevalent in local area networks. In variable-size framing, we need a way to define the end of the frame and the beginning of the next. Historically, two approaches were used for this purpose: a character-oriented approach and a bit-oriented approach.



2.) Apply bit stuffing 1010011111000000111111100.

Bit stuffing is a technique used in data transmission to ensure that special bit patterns (like the flag sequence used to indicate the start or end of a frame) do not appear in the data part of the frame. This is typically done by inserting a '0' after a sequence of five consecutive 1's in the data.

Let's apply bit stuffing to the given binary sequence.

i.) Identify sequences of five consecutive 1's.

ii.) Insert a '0' after each such sequence.

Here's how the sequence tranforms:

i.) Firstly we come across, 10100 ⟶ since there are no consecutive 1's move to the next bits.

ii.) Next we have 11111, which is five consecutive 1's. Add a '0' bit.

Similarly repeat the process. Finally we get,

101001111100000001111011100

PART II

3. i) Write the steps involved in internet checksum algorithm on both Sender and Receiver side. ii) Evaluate checksum value of 1001001110010011 and 1001100001001101 consider 16-bit data segment.

Internet Checksum

Traditionally, the Internet has been using a 16-bit checksum. The sender calculates the checksum by following these steps.

Sender site:

1. The message is divided into 16-bit words.

2. The value of the checksum word is set to O.

3. All words including the checksum are added ushtg one's complement addition.

4. The sum is complemented and becomes the checksum.

5. The checksum is sent with the data.

Receiver site:

1. The message (including checksum) is divided into 16-bit words.

2. All words are added using one's complement addition.

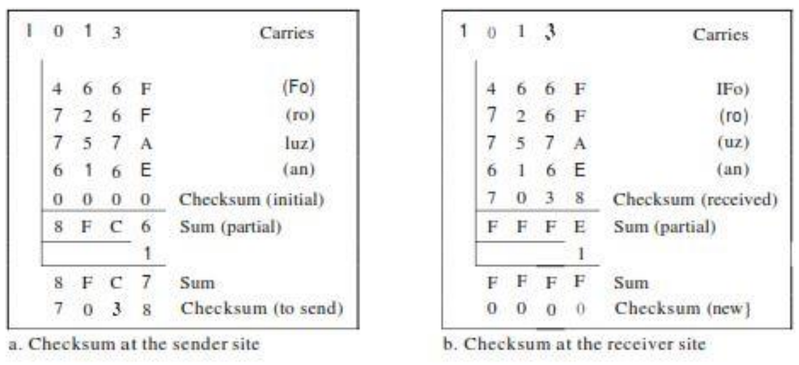
3. The sum is complemented and becomes the new checksum.

4. Ifthe value ofchecksum is 0, the message is accepted; otherwise, it is rejected.

The nature of the checksum (treating words as numbers and adding and comple- menting them) is well-suited for software implementation. Short programs can be written

to calculate the checksum at the receiver site or to check the validity of the message at the receiver site.

Figure 10.25



```
1 0 1 3              Carries          1 0 1 3              Carries

  4 6 6 F            (Fo)               4 6 6 F            IFo)
  7 2 6 F            (ro)               7 2 6 F            (ro)
  7 5 7 A            luz)               7 5 7 A            (uz)
  6 1 6 E            (an)               6 1 6 E            (an)
  0 0 0 0  Checksum (initial)           7 0 3 8  Checksum (received)
  8 F C 6  Sum (partial)                F F F E  Sum (partial)
        1                                     1
  8 F C 7  Sum                          F F F F  Sum
  7 0 3 8  Checksum (to send)           0 0 0 0  Checksum (new)
a. Checksum at the sender site        b. Checksum at the receiver site
```

3.) Evaluate checksum value :-

To evaluate the checksum of two 16-bit data segments, you follow these steps:

i.) Add the two 16-bit binary numbers.
ii.) If there is a carry out of the most significant bit, wrap it around and add it to the least significant bit.
iii.) Complement the result to get the checksum.

Add the two 16-bit binary numbers.

```
     1001001110010011
  +  1001110000100101
  --------------------
    10011011111000000
```

Check for carry out of the most significant bit. The result obtained is a 17-digit number due to the carry. So, we need to wrap ~~it~~ around the carry & add it to the LSB.

Take the carry bit 1 and add it to the least significant 16 bits :

0110111111000000 + 1 = 0110111111000001

Complement the result. So the checksum is,

1001000000111110

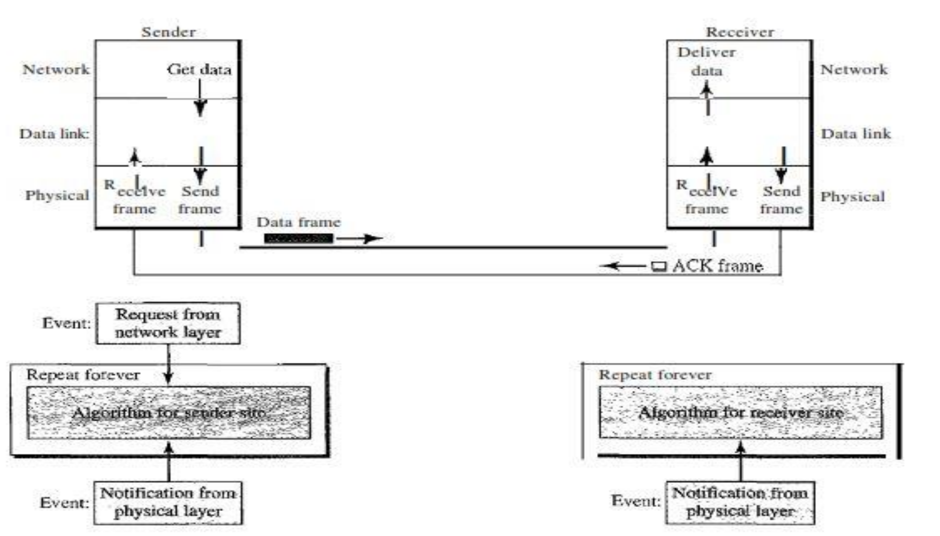
4. Discuss the design of stop and wait protocol for Noiseless Channel. Write down an Algorithm for the same on the sender and receiver side.

Stop-and-Wait Protocol

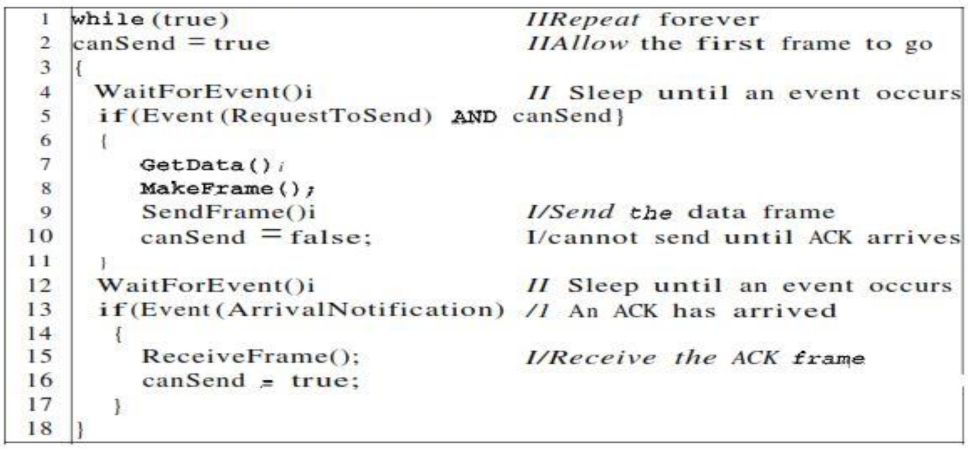
If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use. Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources. This may result in either the discarding of frames or denial of service. To prevent the receiver from becoming overwhelmed with frames, we somehow need to tell the sender to slow down. There must be feedback from the receiver to the sender. The protocol we discuss now is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame.

We still have unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction. We add flow control to our previous protocol. Design Figure 11.8 illustrates the mechanism. Comparing this figure with Figure 11.6, we can see the traffic on the forward channel (from sender to receiver) and the reverse channel. At any time, there is either one data frame on the forward channel or one ACK frame on the reverse channel. We therefore need a half-duplex link.

Figure 11.8 *Design of Stop-and- Wait Protocol*



Algorithm 11.3 *Sender-site algorithm for Stop-and- Wait Protocol*



```
while(true)                          IIRepeat forever
canSend = true                       IIAllow the first frame to go
{
  WaitForEvent()i                    II Sleep until an event occurs
  if(Event(RequestToSend) AND canSend}
  {
     GetData();
     MakeFrame();
     SendFrame()i                    I/Send the data frame
     canSend = false;                I/cannot send until ACK arrives
  }
  WaitForEvent()i                    II Sleep until an event occurs
  if(Event(ArrivalNotification)     /1 An ACK has arrived
   {
     ReceiveFrame();                 I/Receive the ACK frame
     canSend = true;
   }
}
```

PART III

5. Explain the frame Formats of HDLC and PPP protocol in detail.

Frames

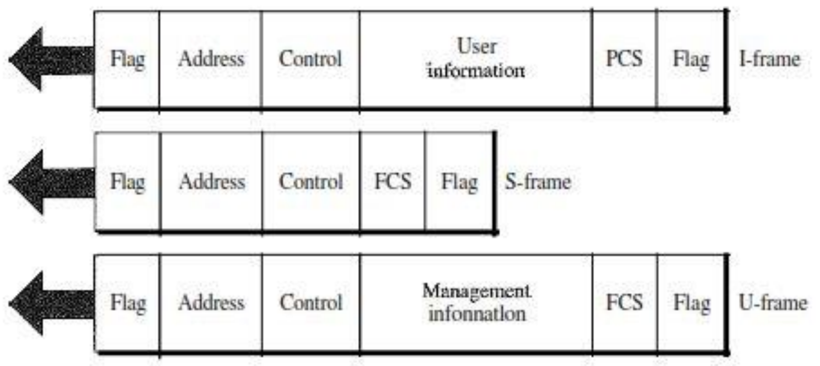
To provide the flexibility necessary to support all the options possible in the modes and configurations just described, HDLC defines three types of frames: information frames(I-frames), supervisory frames (S-frames), and unnumbered frames (V-frames). Each type of frame serves as an envelope for the transmission of a different type of message. I-frames are used to transport user data and control information relating to user data (piggybacking). S-frames are used only to transport control information. V-frames are reserved for system management. Information carried by V-frames is intended for manag-ing the link itself.

Frame Format
Each frame in HDLC may contain up to six fields, as shown in Figure 11.27: a begin-ning flag field, an address field, a control field, an information field, a frame check sequence (FCS) field, and an ending flag field. In multiple-frame transmissions, the ending flag of one frame can serve as the beginning flag of the next frame.

Fields

Let us now discuss the fields and their use in different frame types.
o Flag field. The flag field of an HDLC frame is an 8-bit sequence with the bit pattern 01111110 that identifies both the beginning and the end of a frame and serves as a synchronization pattern for the receiver.

Figure 11.27 *HDLC frames*



o Address field. The second field of an HDLC frame contains the address of the secondary station. If a primary station created the frame, it contains a to address. If a secondary creates the frame, it contains afrom address. An address field can be

1 byte or several bytes long, depending on the needs of the network. One byte can identify up to 128 stations (l bit is used for another purpose). Larger networks require multiple-byte address fields. If the address field is only 1 byte, the last bit is always a
1. If the address is more than 1 byte, all bytes but the last one will end with 0; only the last will end with
1. Ending each intermediate byte with 0 indi-cates to the receiver that there are more address bytes to come.
o Control field. The control field is a 1- or 2-byte segment of the frame used for flow and error control. The interpretation of bits in this field depends on the frame type.
We discuss this field later and describe its format for each frame type.
o Information field. The information field contains the user's data from the net-work layer or management information. Its length can vary from one network to another.
o FCS field. The frame check sequence (FCS) is the HDLC error detection field. It can contain either a 2- or 4-byte ITU-T CRC.

What is HDLC?

High-level Data Link Control (HDLC) could be a synchronous bit-oriented information interface layer convention utilized for transmitting data over point-to-point and multipoint joins. It could be a broadly utilized convention that gives solid and proficient information transmission between organized gadgets.

HDLC operates in two primary modes −

Normal Response Mode (NRM) and Asynchronous Balanced Mode (ABM). In NRM, one gadget called the essential station, controls communication by starting and overseeing information exchange. ABM, on the other hand, permits both gadgets to act as breaks even, with either gadget starting communication or both gadgets having risen to duties.

HDLC frames comprise a header, data field, and a trailer. The header contains control data, such as the address of the goal station, control bits for stream control, and mistake discovery data. The data field carries the real information being transmitted. The trailer contains a cyclic repetition check (CRC) esteem for mistake discovery.

HDLC bolsters different transmission modes, counting full-duplex and half-duplex modes. It moreover gives components for blunder discovery and redress, stream control, and multiplexing different consistent channels over a single physical connection.

The HDLC convention has been broadly embraced in numerous organizing innovations, counting synchronous serial interfaces, Integrated Administrations Computerized Organize (ISDN), and X.25 systems. It serves as an establishment for other conventions such as Cisco's proprietary convention, Point-to-Point Convention (PPP), and subsidiaries like Outline Hand-off.

HDLC utilizes cyclic repetition check (CRC) for mistake discovery, which includes adding a checksum to the outline to confirm information keenness amid transmission. In any case, it does not give built-in blunder redress components.

In spite of the fact that HDLC was initially created for utilization in synchronous serial communication joins it has found far-reaching appropriation in different organized situations.

What is PPP?

Point-to-Point Protocol (PPP) is a Synonyms information interface layer convention utilized for building up a coordinate association between two arrange hubs, regularly over a serial interface. It gives a standard strategy for transmitting information bundles over different physical media, such as serial cables, phone lines, or fiber optic joins.

PPP offers a solid and effective way to set up and keep up a communication connection between two gadgets, permitting them to trade network-layer parcels. It underpins confirmation, blunder location, and multilink capabilities.

PPP employs a Link Control Protocol (LCP) to arrange and design the association parameters, such as verification strategies and organize conventions to be utilized

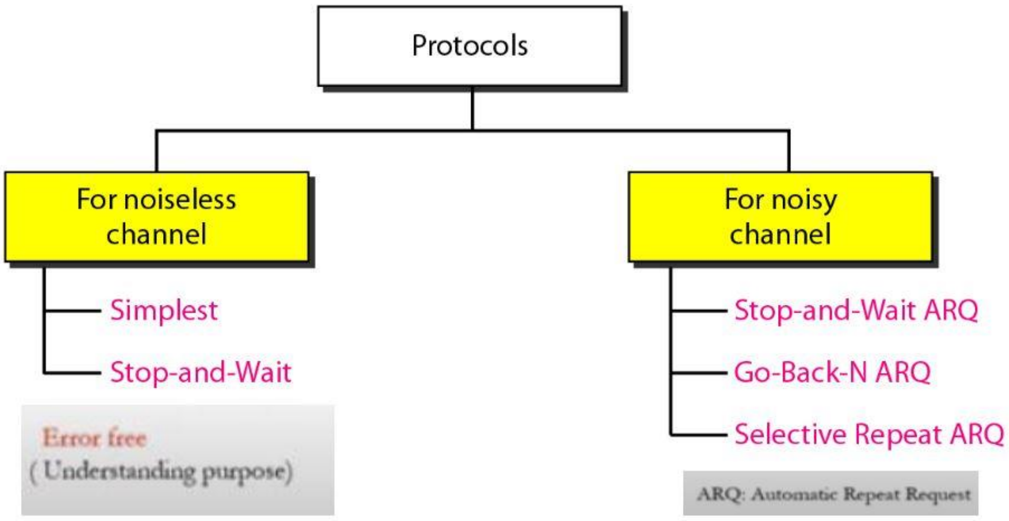
PPP too underpins different confirmation strategies, counting Secret word Confirmation Convention and Challenge Handshake Confirmation Convention, which guarantee secure and confirmed associations between arranged gadgets.

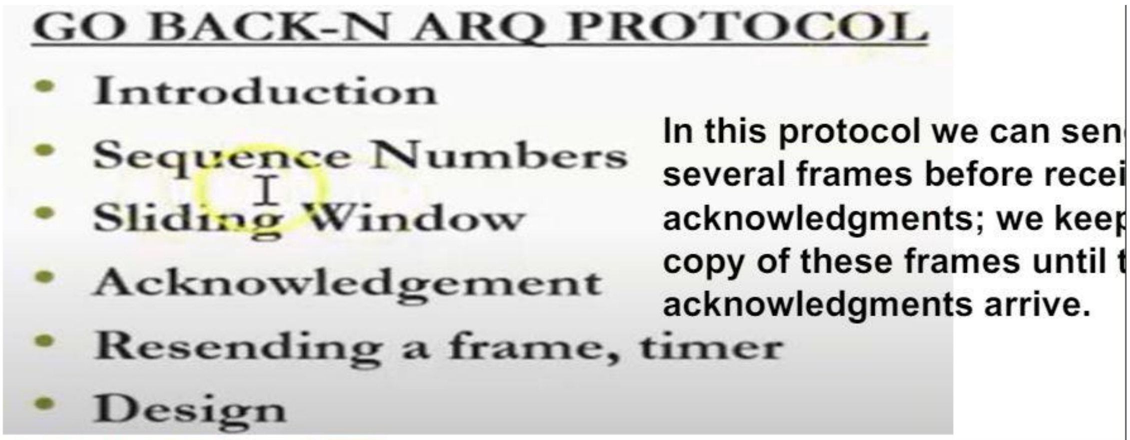
6 Give the Taxonomy of Protocols of Flow and Error Control. Briefly explain Go Back N ARQ protocol with an example.

1. Flow Control :
It is an important function of the Data Link Layer. It refers to a set of procedures that tells the sender how much data it can transmit before waiting for acknowledgment from the receiver.
Purpose of Flow Control :
Any receiving device has a limited speed at which it can process incoming data and also a limited amount of memory to store incoming data. If the source is sending the data at a faster rate than the capacity of the receiver, there is a possibility of the receiver being swamped. The receiver will keep losing some of the frames simply because they are arriving too quickly and the buffer is also getting filled up.
This will generate waste frames on the network. Therefore, the receiving device must have some mechanism to inform the sender to send fewer frames or stop transmission temporarily. In this way, flow control will control the rate of frame transmission to a value that can be handled by the receiver.

The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control. Flow Control Error Control Topics discussed in this section:
Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.

PROTOCOLS Now let us see how the data link layer can combine framing, flow control, and error control to achieve the delivery of data from one node to another. The protocols are normally implemented in software by using one of the common programming languages. To make our discussions language-free, we have written in pseudocode a version of each protocol that concentrates mostly on the procedure instead of delving into the details of language rules. 11.17 Figure 11.5 Taxonomy of protocols discussed in this chapter

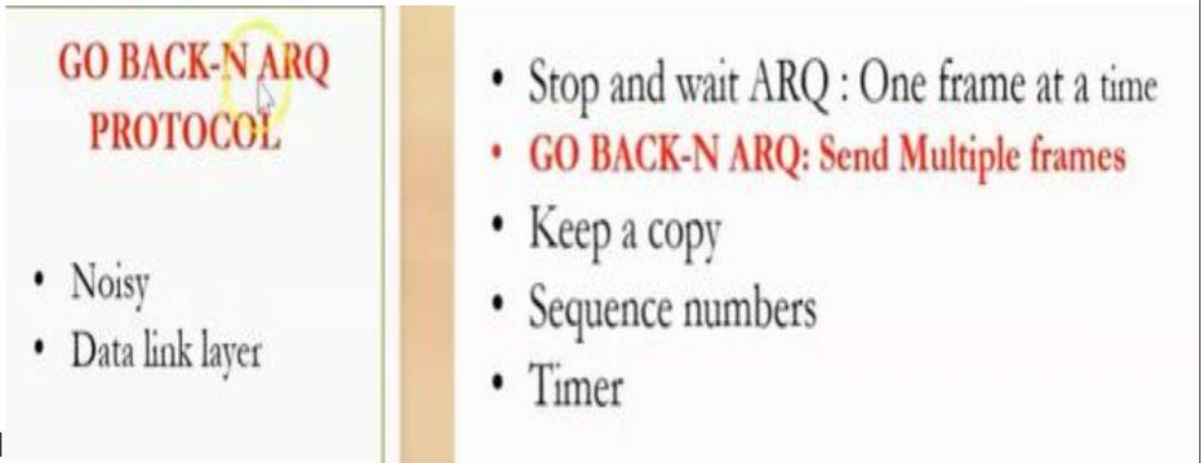




GO BACK-N ARQ PROTOCOL

- Introduction
- Sequence Numbers
- Sliding Window
- Acknowledgement
- Resending a frame, timer
- Design

In this protocol we can sen
several frames before recei
acknowledgments; we keep
copy of these frames until t
acknowledgments arrive.



GO BACK-N ARQ PROTOCOL

- Noisy
- Data link layer

- Stop and wait ARQ : One frame at a time
- GO BACK-N ARQ: Send Multiple frames
- Keep a copy
- Sequence numbers
- Timer

In the Go-Back-N Protocol, the sequence numbers are modulo 2m , where m is the size of the sequence number field in bits

Frames from a sending station are numbered sequentially. However, because we need to include the sequence number of each frame in the header, we need to set a limit. If the header of the frame allows m bits for the sequence number, the sequence numbers range from 0 to 2m - 1.

Go-Back-N Automatic Repeat Request

To improve the efficiency of transmission (filling the pipe), multiple frames must be in

transition while waiting for acknowledgment. In other words, we need to let more than one frame be outstanding to keep the channel busy while the sender is waiting for acknowledgment. In this section, we discuss one protocol that can achieve this goal; in the next section, we discuss a second.
The first is called Go-Back-N Automatic Repeat Request (the rationale for the name will become clear later). In this protocol we can send several frames before

receiving acknowledgments; we keep a copy of these frames until the acknowledg-ments arrive.

Sequence Numbers
Frames from a sending station are numbered sequentially. However, because we need to include the sequence number of each frame in the header, we need to set a limit. If the header of the frame allows m bits for the sequence number, the sequence numbers range from 0 to 2m - 1. For example, if m is 4, the only sequence numbers are 0

through 15 inclusive. However, we can repeat the sequence. So the sequence num-bers are

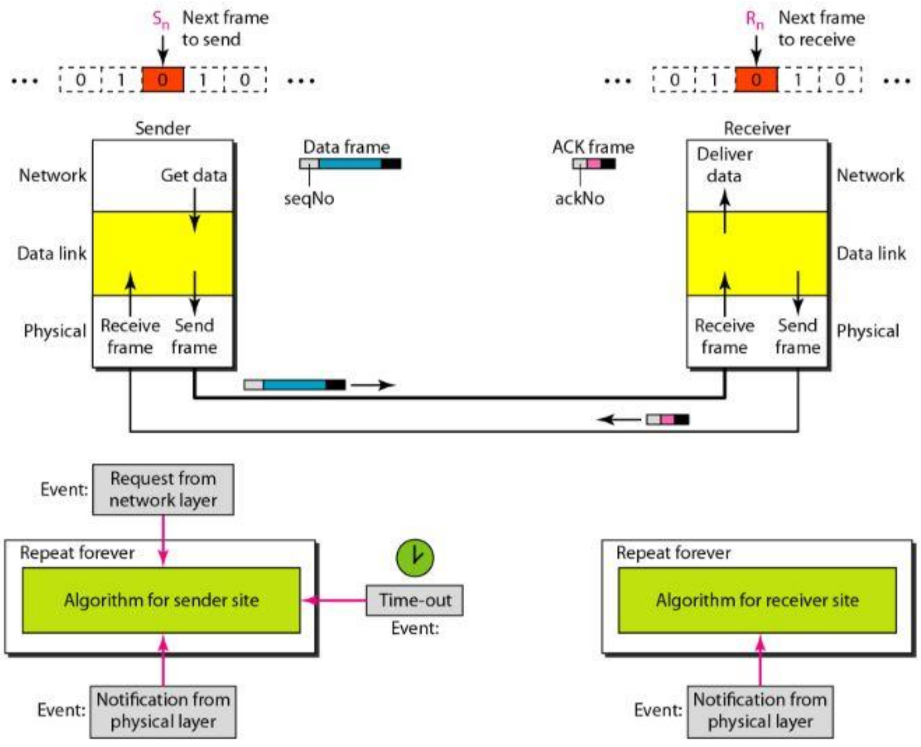
0, 1,2,3,4,5,6, 7,8,9, 10, 11, 12, 13, 14, 15,0, 1,2,3,4,5,6,7,8,9,10, 11, ...
In other words, the sequence numbers are modulo-2m.

PART IV

7. Using 5-bit Sequence Number, what is the maximum size of the Send and Receive windows for each of the following Protocols?
a) Stop and Wait ARQ
b) Selective Repeat ARQ

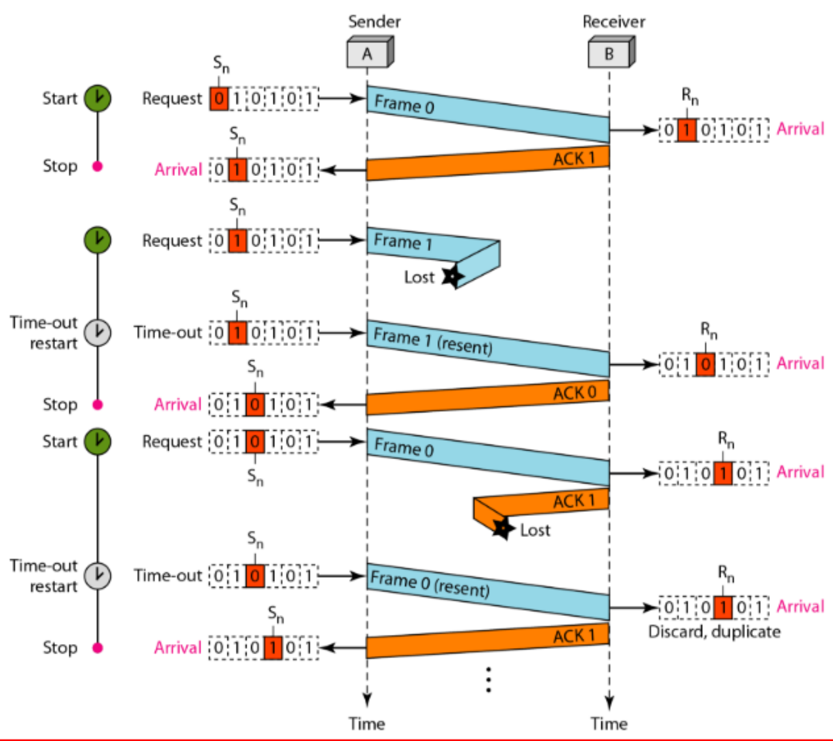
The sending device keeps a copy of the last frame transmitted until it receives an acknowledgment for that frame. A data frames uses a seqNo (sequence number); an ACK frame uses an ackNo (acknowledgment number). The sender has a control variable, which we call Sn (sender, next frame to send), that holds the sequence number for the next frame to be sent (0 or 1). The receiver has a control variable, which we call Rn (receiver, next frame expected), that holds the number of the next frame expected. When a frame is sent, the value of Sn is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. When a frame is received, the value of Rn is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. Three events can happen at the sender site; one event can happen at the receiver site

Figure 11.10 ***Design of the Stop-and-Wait ARQ Protocol***



The sending device keeps a copy of the last frame transmitted until it receives an acknowledgment for that frame. A data frames uses a seqNo (sequence number); an ACK frame uses an ackNo (acknowledgment number). The sender has a control variable, which we call Sn (sender, next frame to send), that holds the sequence number for the next frame to be sent (0 or 1). The receiver has a control variable, which we call Rn (receiver, next frame expected), that holds the number of the next frame expected. When a frame is sent, the value of Sn is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. When a frame is received, the value of Rn is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. Three events can happen at the sender site; one event can happen at the receiver site 11.42 Algorithm 11.5

Figure 11.11 *Flow diagram for Example 11.3*

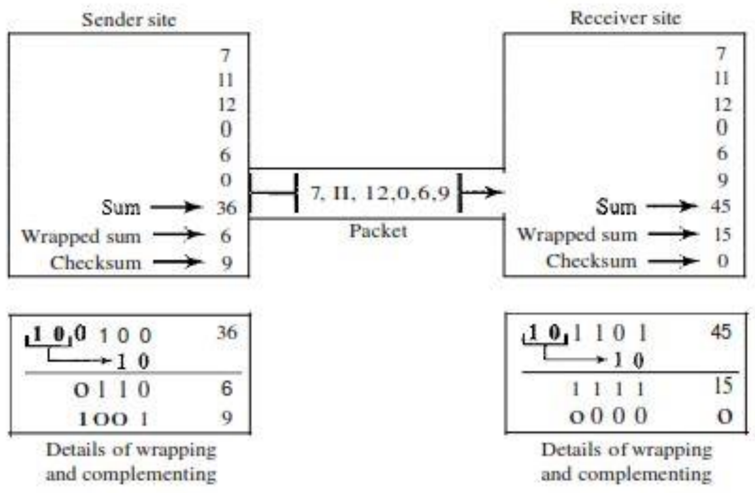


OR

8. Suppose our data is a list of 4-bit no’s that we want to send to a destination. If the set of number is (7,11,12,0,6) then explain the complete checksum process at both sender and receiver side.

The last error detection method we discuss here is called the checksum. The checksum is used in the Internet by several protocols although not at the data link layer. However, we briefly discuss it here to complete our discussion on error checking.
Like linear and cyclic codes, the checksum is based on the concept of redundancy.
Several protocols still use the checksum for error detection as we will see in future chapters, although the tendency is to replace it with a CRC. This means that the CRC is also used in layers other than the data link layer.

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 are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.



Internet Checksum

Traditionally, the Internet has been using a 16-bit checksum. The sender calculates the checksum by following these steps.

Sender site:
1. The message is divided into 16-bit words.
2. The value of the checksum word is set to O.
3. All words including the checksum are added ushtg one's complement addition.
4. The sum is complemented and becomes the checksum.
5. The checksum is sent with the data.

The receiver uses the following steps for error detection.

Receiver site:
1. The message (including checksum) is divided into 16-bit words.
2. All words are added using one's complement addition.
3. The sum is complemented and becomes the new checksum.
4. If the value of checksum is 0, the message is accepted; otherwise, it is rejected.

The nature of the checksum (treating words as numbers and adding and comple-menting them) is well-suited for software implementation. Short programs can be written
to calculate the checksum at the receiver site or to check the validity
of the message at the receiver site.

PART V

9. What do you understand by Block Coding Schemes? Explain Structure of Encoder and Decoder in Error Correction.

Block Coding

We need redundancy to ensure synchronization and to provide some kind of inherent error detecting. Block coding can give us this redundancy and improve the perfor-mance of line coding. In general, block coding changes a block of m bits into a block of n bits, where n is larger than m. Block coding is referred to as an mB/nB encoding technique.

Block coding is normally referred to as mBlnB coding;
it replaces each `m~bit` group with an `n~bit` group.

The slash in block encoding (for example, 4B/5B) distinguishes block encoding from multilevel encoding (for example, 8B6T), which is written without a slash. Block coding normally involves three steps: division, substitution, and combination.
In the division step, a sequence of bits is divided into groups of m bits. For example, in 4B/5B encoding, the original bit sequence is divided into 4-bit groups. The heart of block cod-ing is the substitution step. In this step, we substitute an m-bit group for an n-bit group.
For example, in 4B/5B encoding we substitute a 4-bit code for a 5-bit group. Finally, the n-bit groups are combined together to form a stream. The new stream has more bits than the original bits. Figure 4.14 shows the procedure.

Figure 4.14 *Block coding concept*

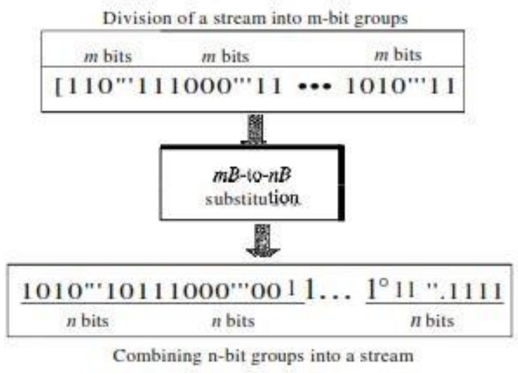


Figure 4.15 *Using block coding 4B/5B with NRZ-I line coding scheme*

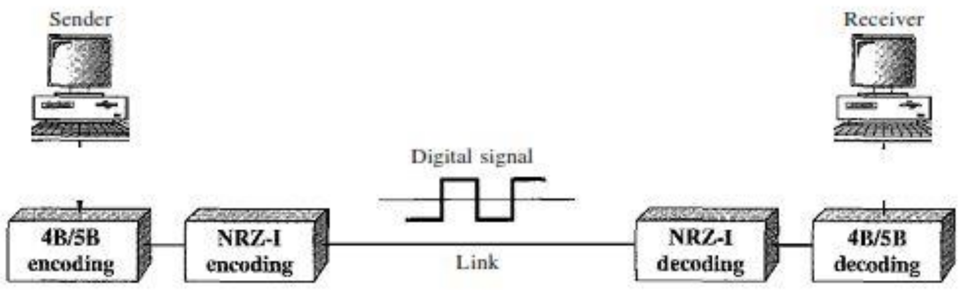


Table 4.2 *4B/5B mapping codes*

Data Sequence	*Encoded Sequence*	*Control Sequence*	*Encoded Sequence*
0000	11110	Q (Quiet)	00000
0001	01001	I (Idle)	11111
0010	10100	H (Halt)	00100
0011	10101	J (Start delimiter)	11000
0100	01010	K (Start delimiter)	10001
0101	01011	T (End delimiter)	01101

Table 4.2 *4B/5B mapping codes (continued)*

Data Sequence	*Encoded Sequence*	*Control Sequence*	*Encoded Sequence*
0110	01110	S (Set)	11001
0111	01111	R (Reset)	00111
1000	10010		
1001	10011		
1010	10110		
1011	10111		
1100	11 010		
1101	11011		
1110	11100		
1111	11101		

Figure 4.16 *Substitution in 48/5B block coding*

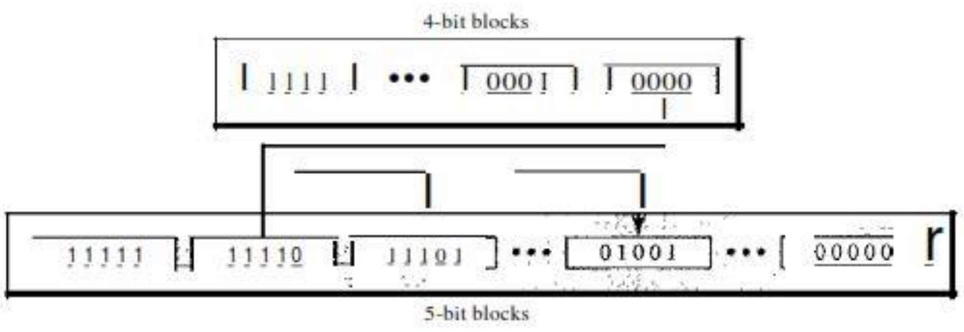


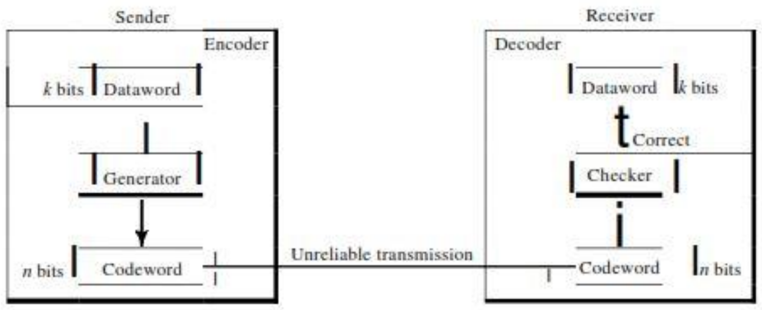
Figure 4.16 shows an example of substitution in 4B/5B coding. 4B/5B encoding solves the problem of synchronization and overcomes one of the deficiencies of NRZ-1. However, we need to remember that it increases the signal rate of NRZ-1. The redundant bits add 20 percent more baud. Still, the result is less than the biphase scheme which has a signal rate of 2 times that of NRZ-1. However, 4B/5B block encoding does not solve the DC component problem of NRZ-1. If a DC component is unacceptable, we need to use biphase or bipolar encoding.

Example 4.5
We need to send data at a 1-Mbps rate. What is the minimum required bandwidth, using a combi-nation of 4B/5B and NRZ-I or Manchester coding?
Solution
First 4B/5B block coding increases the bit rate to 1.25 Mbps. The minimum bandwidth using NRZ-I is NI2 or 625 kHz. The Manchester scheme needs a minimum bandwidth of 1 MHz. The first choice needs a lower bandwidth, but has a DC component problem; the second choice needs a higher bandwidth, but does not have a DC component problem.

Error Correction

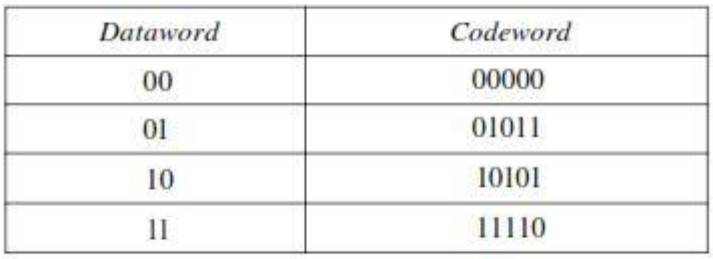
As we said before, error correction is much more difficult than error detection. In error detection, the receiver needs to know only that the received codeword is invalid; in error correction the receiver needs to find (or guess) the original codeword sent.
We can say that we need more redundant bits for error correction than for error detection. Figure 10.7 shows the role of block coding in error correction. We can see that the idea is the same as error detection but the checker functions are much more complex.

Figure 10.7 *Structure ofencoder 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. Again, later we will show how we chose the redundant bits. For the moment let us concentrate on the error correction concept. Table 10.2 shows the datawords and codewords.
Assume the dataword is 01. The sender consults the table (or uses an algorithm) to create the codeword 01011. The codeword is corrupted during transmission, and 01001 is received (error the second bit from the right). First, the receiver finds that the received codeword is not in the table.
This means an error has occurred. (Detection must come before correction.) The receiver, assuming that there is only 1 bit corrupted, uses the following strategy to guess the correct dataword.

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



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

I. 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.

OR

10 Explain Linear Block Codes with diagram.

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 not as widespread because their structure makes theoretical analysis and implementation difficult. We therefore con-centrate on linear block codes.
The formal definition of linear block codes requires the knowledge of abstract algebra (particularly Galois fields), which is beyond the scope of this book. We therefore give an informal definition. For our purposes, a linear block code is a code in which the exclusive OR (addition modulo-2) of two valid codewords creates another valid codeword.

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.

Minimum Distance for Linear Block Codes

It is simple to find the minimum Hamming distance for a linear block code. The mini-
mum Hamming distance is the number of Is in the nonzero valid codeword with the
smallest number of Is.

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

=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 min =3.

Some Linear Block Codes

Let us now show some linear block codes. These codes are trivial because we can easily
find the encoding and decoding algorithms and check their performances.

Simple Parity-Check Code Perhaps the most familiar error-detecting code is the simple parity-check 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 Is in the codeword even. Although some implementations specify an odd number of Is, we discuss the even case.

The minimum Hamming distance for this category is d =2, which means that
the code is a single-bit error-detecting code; it cannot correct any error.

A simple parity-check code is a single-bit error-detecting

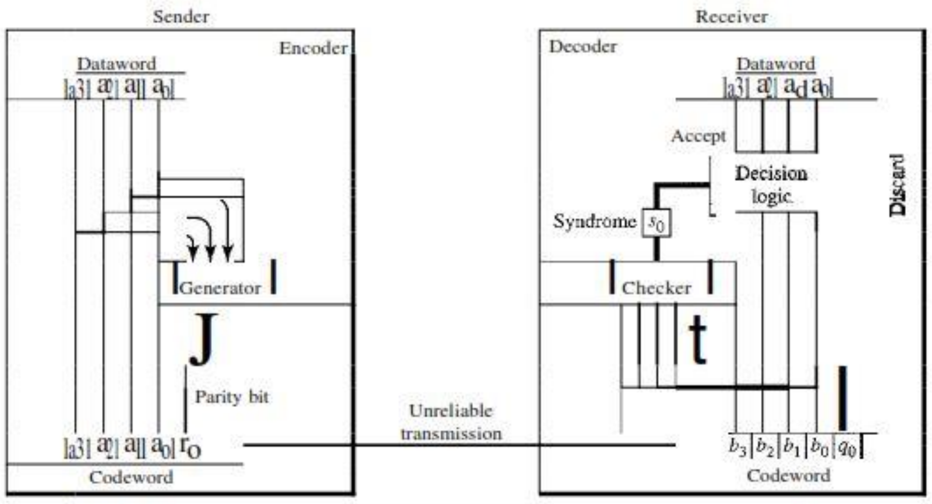
code in which n =k + 1 with dminmin=2.

Our first code (Table 10.1) is a parity-check code with k -= 2 and n =3. The code in Table 10.3 is also a parity-check code with k =4 and n =5. 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 (ao, aI' a2' and a3) and generates a parity bit roo 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.

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

Figure 10.10 *Encoder and decoderfor simple parity-check code*

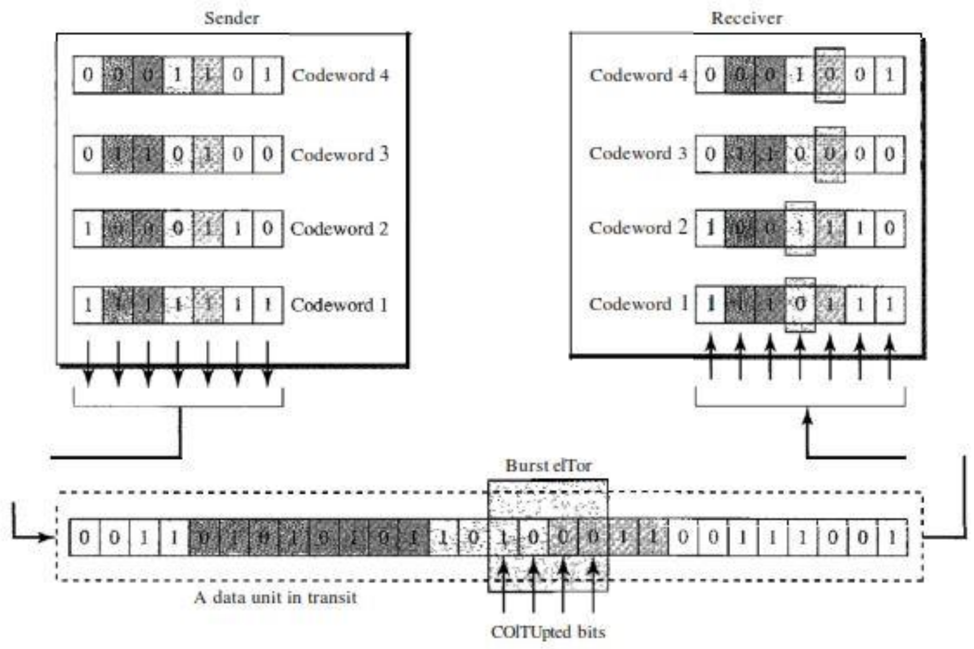


This is normally done by adding the 4 bits of the dataword (modulo-2); the result is the parity bit. In other words, If the number of 1s is even, the result is 0; if the number of 1s is odd, the result is 1.
In both cases, the total number of 1s in the codeword is even.
The sender sends the codeword which may be corrupted during transmission. The receiver receives a 5-bit word. The checker at the receiver does the same thing as the generator in the sender with one exception: 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.

Performance

A Hamming code can only correct a single error or detect a double error. However, there is a way to make it detect a burst error, as shown in Figure 10.13. The key is to split a burst error between several codewords, one error for each codeword. In data communications, we normally send a packet or a frame of data. To make the Hamming code respond to a burst error of size N, we need to make N codewords out of our frame. Then, instead of sending one codeword at a time, we arrange the codewords in a table and send the bits in the table a column at a time. In Figure 10.13, the bits are sent column by column (from the left). In each column, the bits are sent from the bottom to the top. In this way, a frame is made out of the four codewords and sent to the receiver. Figure 10.13 shows

Figure 10.13 *Burst error correction using Hamming code*



that when a burst error of size 4 corrupts the frame, only 1 bit from each codeword is corrupted. The corrupted bit in each codeword can then easily be corrected at the receiver.