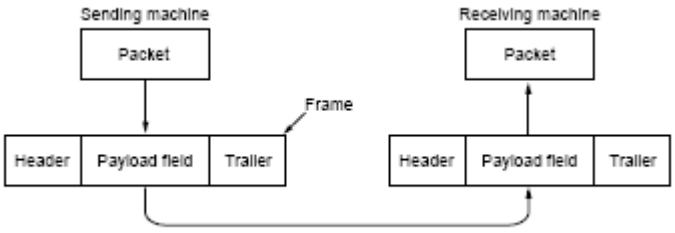
3. DATA LINK LAYER

Data link layer deals with mechanisms for achieving reliable, efficient communication between two adjacent machines at the data link layer. Here adjacent means that two machines are physically connected by a communication channel that acts like a wire. Unfortunately communication lines make errors occasionally.

Design Issues: This layer has a number of specific functions to carry. These functions are discussed below

- ✓ To provide well defined services to the network layer
- ✓ To find the physical address of the communicating parties. **(Physical Address)**
- ✓ To determine how the bits of physical layer are grouped into frames **(framing)**
- ✓ To deal with transmission errors **(Error Control**)
- ✓ To control the flow of frames to recover slow receivers from fast senders. **(Flow control)**

To accomplish these goals, the data link layer receives packets from network layer and encapsulates them into frames for transmission. Each frame contains a frame header, a payload field (DATA) and a trailer. Frame management is the heart of the data link layer.

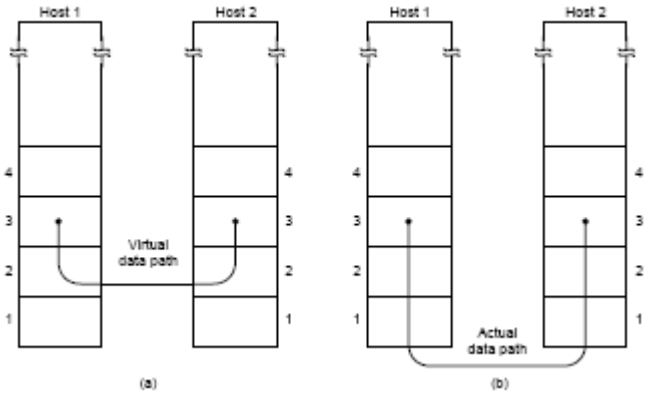


1. Services provided to the Network Layer:

The major service is transferring data from the network layer on the source machine to the network layer on the destination machine. The following figure a shows the virtual path. It provides direct communication from source data link layer to the destination data link layer. But this is not possible. To achieve this, we should have an actual path like in the second figure to exchange data in between source data link layer to the destination data link layer.

The data link layer offers many services. These services can vary from system to system. They are

- ✓ Unacknowledged connectionless service
- ✓ Acknowledged connectionless service
- ✓ Acknowledged connection oriented service



Virtual Communication Actual communication

Unacknowledged Connectionless Service: In this case source distribute independent frames to the destination. Destination need not acknowledge the received frames. No connection is established before transmission or is not released after transmission. If a frame is lost due to noise on the line, no attempt made to recover it in the data link layer. This type of service is appropriate when the error rate is very low. So recovery is left to upper layers.
Ex: real time traffic such as speech. In this, late data is worse than bad data. Most LANs use this service in data link layer

Acknowledged Connectionless service: This is the next type of service in terms of reliability. Here also we do not have any connection establishments or terminations. But each frame sent by source is individually acknowledged. So that, the sender knows whether, the frame has arrived safely or not. If the frame has not arrived within a specified time interval, it can be retransmitted again.
Ex: used for unreliable channels like wireless systems.

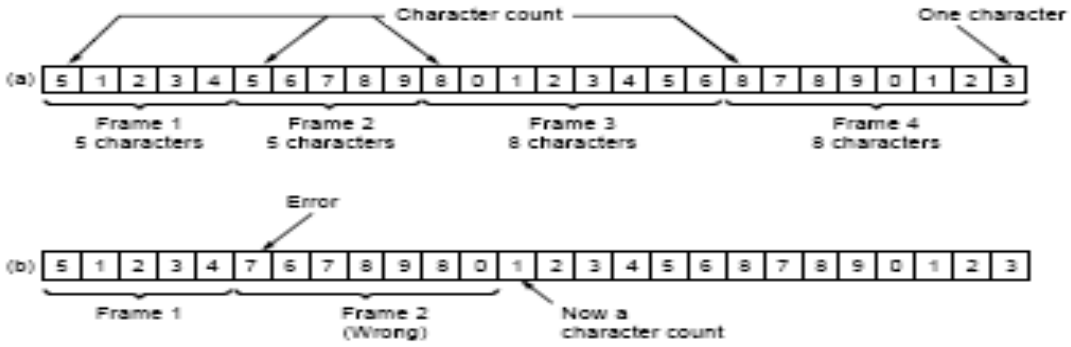
Acknowledged Connection Oriented Service: Providing acknowledgements in data link layer is not required, but it is required to optimize the traffic only. Why because if the frames are not acknowledged, sender does not know either frame has arrived safely or not. So that it will try to retransmit the frame again. That is how it creates lot of traffic. When we prefer this service, all transfers have 3 distinct phases. In the first phase connection is established in between source and destination. The second phase is data transmission phase. So a frame sent from one end to the other end. In the last phase connection is released.

Framing: It defines how the bits of physical layer are grouped into frames. A frame is contiguous no of bits. Physical layer accepts a raw bit stream and deliver it to the destination. This bit stream is not guaranteed to be error free. The number of bits received may be less than, equal to, or more than the number of bits transmitted. So it is up to the data link layer to detect and correct errors.

The general approach is breaking the bit stream into discrete frames and calculating the checksum for each frame. After receiving the frame by destination they also calculate checksum on received data. If the newly calculated checksum and transmitted checksum are different then an error has occurred in so and so frame. Here we have 4 different method for framing.

Framing Methods:

1. Character Count: This method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count, it knows how many characters are there in a frame and where the frame is going to end. This technique is shown in the following figure, for four frames of sizes 5,5,8, and 9 characters respectively.



The major difficult with this method is the count can be garbled by a transmission error. For example if the character count of 5 in the second frame becomes a 7 in the following figure. The destination will get out of synchronization and is unable to locate the start of the nest frames even if the checksum is wrong and the destination finds that the frame is bad.

2. Character Stuffing: Character stuffing is applied at the starting and ending characters. This method gets around resynchronization after an error by having each frame start with the ASCII character sequence DLE STX and end with the sequence DLE ETX. Where DLE is Data Link Escape and STX is the Start of the text.

Ex: DLE STX A DLE B DLE ETX (data sent by the network layer)
DLE STX A DLE DLE B DLE ETX (Character stuffed by Data Link Layer)

DLE STX A DLE B DLE ETX (data passed to the n/w layer at the receiver side)

3. Bit Stuffing: The bit stuffing is analogous to character stuffing, in which a DLE is stuffed into the outgoing character stream before DLE in the data. Here in this method when we have 5 consecutive 1's by any 1 bit or 0 bit. We stuff the 0 bit. That is if there are 5 consecutive 1's we stuff 0 bit.

At the receiver side, when they receive 5 consecutive 1's or 5 consecutive 0's then the receiver de stuffs the following 0 bit or 1 bit respectively.

a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 (Original data)
b) 0 1 1 0 1 1 1 1 1 **0** 1 1 1 1 1 **0** 1 1 1 1 1 **0** 1 0 0 1 0 (stuffed data) Bold 0's are stuffed bits
c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 (de stuffing at The receiver side)

4. Physical Layer Coding Violations: It is applicable to networks in which the encoding on the physical medium contains some redundancy. For example some LANs encode a 1 bit as 2 physical bits. Normally a 1 bit is a high-low pair and a 0 bit is a low-high pair. The combinations high-high and low-low are not used for data. This means that every data bit has a transition in the middle, making it easy for the receiver to locate the bit boundaries.

Error Control:

Networks must be able to transfer data from one device to another device with complete accuracy. A system can nit guarantee that the data received by one device is identical to the transmitted by another device. At any time data may be corrupted during transmission from source to destination. Some times it may happen that only some part of the message is altered in the original message. Many factors influence this issue. But a reliable system must have a mechanism to detect and correct such errors. Error detection and correction are implemented either at the data link layer or at the transport layer.

Types of Errors: Whenever an electromagnetic signal flows from one point to other, it is subject to unpredictable interference from heat, magnetism, and other, it is subject to unpredictable interference from heat, magnetism, and other forms of electricity. This interference can change shape or timing of the signal. If it is single bit error, a 0 is changed to 1 or a 1 bit is changed to 0. But in a burst error multiple bits are changed.

Single Bit Error: It means that only one bit of a data unit is changed from 1 to 0 or from 0 to 1. To understand this, here we discuss an example by taking an ASCII character 1000001 (Letter A).

1	0	0	0	**0**	0	1

1	0	0	0	**1**	0	1

Here a 0 is changed to 1. However a single bit error can happen if we are sending data using parallel transmission. For example if 7 wires are used to send all of the 7 bits at the same time and one of the wires is noisy, then one bit can be corrupted in each byte.

Burst Error: It means that two or more bits in the data unit have changed from 1 to 0 or from 0 to 1. The length of the burst is measured from the first corrupted bit to the last corrupted but. Some bits in between them may not have corrupted. It is mostly happen in serial transmission. Burst length is calculated from initial corrupted bit to the final corrupted bit though we have some uncorrupted bits. This length is known as Hamming Distance.

Sent

0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

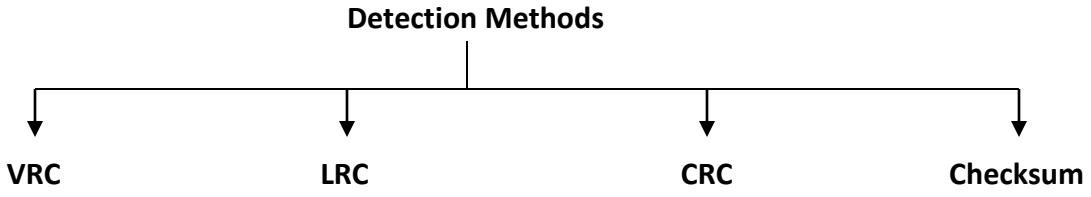
Received

0	1	0	1	1	1	0	1	0	1	0	0	0	0	1	1

Error Detection: One of the error detection mechanism is to send data twice every time. The receiver then does bit by bit comparison between the two versions of the data. If he identifies any difference, he assumes that there is an error and applies some appropriate correction mechanism. But it is not the acceptable mechanism. Why because, it doubles the transmission time and takes lot of time to compare each bit in the two versions of the data. Here two versions means the data sent by the sender and the data received by the receiver.

Therefore the concept of adding extra information during the transmission is for the purpose of having a good comparison only. But instead of repeating the entire data, a small group of bits may be appended to the end of each unit. This technique is called redundancy. These extra bits are redundant to the information for detecting errors. These bits are discarded after checking the accuracy of the data. Four types of redundancy checks are used in data communication. Vertical Redundancy Check (VRC), Longitudinal Redundancy Checks (LRC), Cyclic Redundancy Check (CRC), are normally implemented in the physical layer for the use in data link layer. The fourth type check sum is used by upper layers only.

Error Detecting Codes:



Vertical Redundancy Check: It is also known as a parity check. In this method a redundant bit called a parity bit is appended to every data unit so that the total number of 1's in the data becomes even.

Suppose if we want to transmit the binary data unit 1100001 (ASCII a), the number of bits in a given bit stream is 3 that is an odd number. Before transmitting we send the data to a parity generator. The parity generator counts the number of 1's and then appends a parity bit (here in this case, the parity bit 1 is added to make the number of 1's 4) at the end to the data. Now the system transmits the entire expanded data to the destination.

When it reaches to the destination, the receiver puts all 8 bits through an even parity checking. If the number of 1's are odd, then the receiver knows an error has occurred into the data somewhere and therefore rejects whole data unit. Some systems may use odd parity concept also.

Ex: The sender wants to send the word **"Ether"** to the destination.

For this, first he takes the ASCII value of Ether

E	t	h	e	r
1000101	1110100	1101000	1100101	1110010

Afterwards sender calculates parity bit and append it at the least significant bit.

10001011 11101000 11010001 11001010 11100100

If the receiver receives the same information as in above he accepts the frame otherwise he discards it. For example if he receives the following data,

10011011 11001000 11011001 11001010 11100000

Upon receiving the above frame, receiver calculates again the even parity. Here even parity is not maintained that is why the data is discarded. All corrupted bits are presented as in italic font.

Performance: VRC can detect all single bit errors. It can also detect burst errors, if the total number of bits corrupted is odd. That means if we have data 1000111011 where the total number of 1's is 6including parity bit. If any 3 bits will change the resulting parity will be odd. So error will be detected. But suppose if 4 bits change, the resulting parity is even so u can not detect the error. That is why it can detect only odd number of burst errors. It fails in detecting when even number of bits corrupted.

Longitudinal Redundancy Check: In this method a block of bits are organized into rows and columns. For example instead of sending a block of 32 bits, we convert them into a table of 4 rows and 8 columns. We then calculate a parity bit for each column and create a new row of 8 bits, which are the parity bits for the entire block. The first parity bit in the 5th row is calculated based on all first column bits, 2nd parity bit is calculated

based on all 2nd column bits, and so on. At the last we attach all 8 parity bits to the original data and sent them to the receiver.

Ex: original data is 11100111 11011101 00111001 10101001

Arrange all the blocks into rows and columns

```
11100111  row 1
11011101  row 2
00111001  row 3
10101001  row 4
-----------------------------
10101010  LRC
```

Now pad this LRC to the original data and send it to the receiver.

That is, 11100111 110111101 00111001 10101001 10101010 (0riginal data + LRC)

Suppose if the above block is send and however, it is hit by a burst of noise length 8 and some bits are corrupted.

11*0*01*1*11 *0*1*00*0101 00111001 10101001 10101010 given LRC by source when the receiver checks and calculates the LRC,

```
11001111
01000101
00111001
10101001
--------
00011010 calculated LRC by receiver
```

Some of the bits (bold) are not matching to the given LRC. Some of the bits do not follow even parity concept, so the whole block is discarded.

Performance: LRC increases the performance in detecting burst errors. As in the above example, an LRC of n bits can easily detect a burst error of n bits. A burst error of more than n bits is also detected by LRC but with high probability. But here we have a difficulty if two bits in one block are damaged, then this LRC checker will not detect errors.

Ex:

```
11110000        01110001
11000011        01000010
--------        --------
00110011        00110011
```

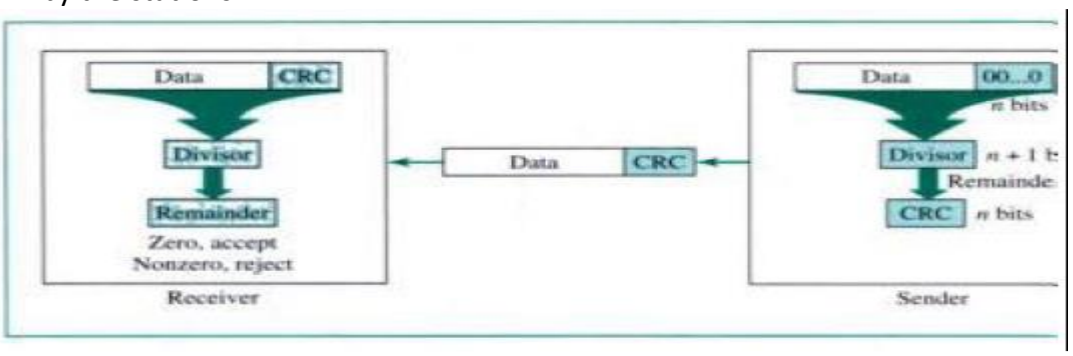
In both the cases we got the same LRC. In the second case the first and last bits of two blocks have changed that is why we are not able to detect the errors.

Cyclic Redundancy Check: Unlike VRC, and LRC which are based on binary addition, it is based on binary division. So here we take the remainder as the redundant information of the data. This remainder is appended to the end of the data and is sent to the receiver. Upon receiving the incoming data unit is divided by the same value. If he receives no remainder that is 0 data will be accepted. A remainder indicated that the data has been changed during transmission and therefore it is rejected.

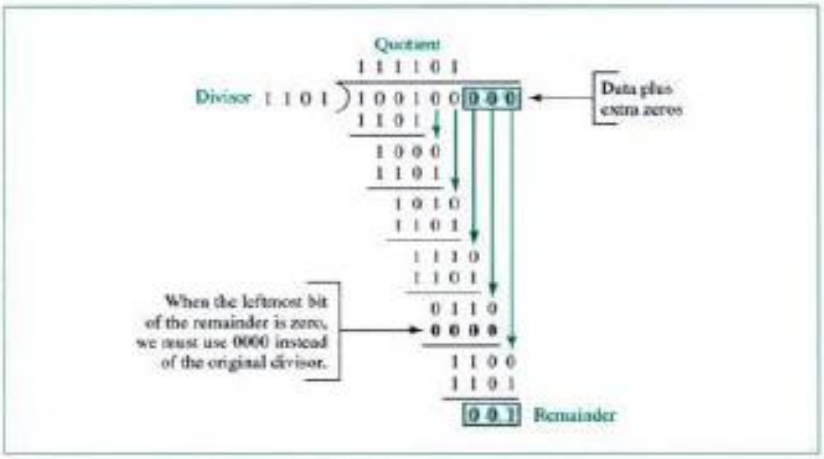
When this method is used both sender and receiver must agree on generator polynomial G(x). Both the high order and low order bits of the generator must be 1. to compute CRC for a frame with m number of bits, the frame must be longer than the generator polynomial.

CRC Algorithm:

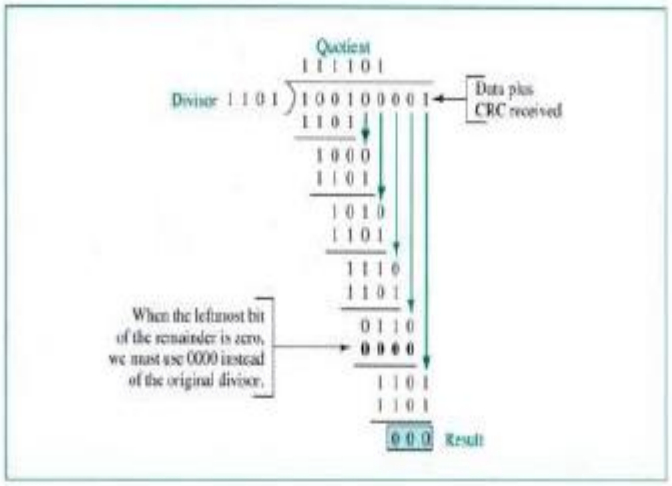
1. Let r be the degree of G(x). Append r zero bits to the low order end of the frame. So it now contains m+r bits.
2. Divide the bit string corresponding to G(x) using modulo 2 division
3. Subtract the remainder from the bit string using modulo 2 subtractions. The result is the check summed frame to be transmitted.
4. After receiving the data, opponent applies same function with the same generator polynomial. If he gets a remainder as zero, then it means that there is nor error in transmission. Other wise there is an error and the frame is discarded by the stations.



CRC Generator: Calculate the CRC for the following frame 100100 with the generator polynomial 1101.



CRC Checker: It is exactly like the generator. After receiving the data, it applies the same modulo 2 division. If the remainder is all 0's the CRC is dropped and the data is accepted, otherwise it is discarded.



Polynomials: The CRC generator is most often represented with an algebraic polynomial. The polynomial format is useful for two reasons. The relationship of a polynomial to its corresponding binary representation is shown in figure.

Performance: CRC is a very effective error detection method. If the divisor is chosen according to the previously mentioned rules.

a. CRC can detect all burst errors that affect an odd number of bits.
b. CRC can detect all burst errors of length less than or equal to the degree of the polynomial.
c. CRC can detect with a very high probability burst errors of length greater than the degree of the polynomial.

Checksum: This is the error detection method used by upper layer protocols. Checksum is also based on the concept of redundancy.

Checksum Generator: It sub divides the data into equal segments of n bits (usually 16). These segments are added together using 1's complement arithmetic in such a way that the total is also n number of bits. This sum is then complemented and appended to the end of the original data as redundant information by using a checksum field. Now this extended data unit is transmitted across the network.

Error Correction: The mechanisms that we have discussed so far are only used to detect errors. Error correction can be handled in two ways.
Case- 1: When an error is discovered, the receiver can request the sender to retransmit the entire data unit.
Case- 2: In other words, a receiver can use an error-correcting code, which automatically corrects certain errors.

Theoretically, it is possible to correct any binary code errors automatically. Error correcting codes, however, are more sophisticated than error detecting codes and require more redundancy bits. The number of bits required to correct a multiple bit or

burst error is so high that in most cases it is inefficient to do so. For this reason, most error correction is limited to one, two, or three bit errors.

Single-Bit Error Correction: single bit errors can be detected by the addition of a redundant bit to the data unit. A single additional bit can detect single-bit errors in any sequence of bits because it distinguishes between two conditions: error or no error. A bit has two states (0 and 1). These two states are sufficient for this level of detection.

But if we want to correct and detect errors, these two states are not sufficient to correct. They are enough to detect the errors only. Why because the secret of error correction is to find the location (position) of the error bits or invalid bits.

For example to correct an error in a single bit error in ASCII character, the error correction code must find which of the seven bits have changed. Therefore we have to distinguish eight different states:

No error, error in position 1, error in position 2, … , error in position 7. To do this we need 3 redundant bits to show all eight states from 000 to 111.

But an error may happen at the redundant bits also.

Redundancy Bits : To calculate the number of redundancy bits(r) required to correct a given number of data bits (m), we must find a relationship between m and r. The length of the resulting code is m+r.

If the total number of bits in a transmittable unit is m+r then r must be able to indicate at least m+r+1 different states. Of these, one state means no error and m+r states indicate the location of an error in each of the m+r positions. So m+r+1 states must be identified by r bits; and r bits can indicate 2^r different states. Therefore 2^r must be equal to or greater than m+r+1.

$$2^r >= m + r + 1$$

Number of Data Bits m	*Number of Redundancy Bits* r	*Total Bits* $m+r$
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11

Hamming Codes: To find the position of the error, we use Hamming codes. This method was developed by J.W. Hamming. The Hamming code can be applied to data units of any length and uses the above relationship between data and redundant bits. According to this, a 7 bit ASCCI code requires 4 redundant bits that can be added to the original data unit. These redundant bits (r) are paced in positions 1, 2, 4, and 8 and in the remaining positions we have the actual data bits (d).

1	2	3	4	5	6	7	8	9	10	11
R1	R2	D1	R4	D2	D3	R8	D4	D5	D6	D7

Position of redundant bits in Hamming Code

In the Hamming Code, each r bit is the VRC bit. That is we get r1, r2, r4, and r8 by doing finding VRC on the data units.

r 1 : 1, 3, 5, 7, 9, 11
r 2 : 2, 3, 6, 7, 10, 11
r 4 : 4, 5, 6, 7
r 8 : 8, 9, 10, 11

That is r1 is calculated with bit positions whose binary representation includes a 1 bit in the right most position. The r2 bit is calculated with bit positions whose binary representation includes a 1 in the second position, r4 is calculated with bit positions whose binary representation includes a 1 bit in the third position, and r8 is calculated with bit positions whose binary representation includes a 1 bit in the 4th position, and so on.

Hamming code Implementation for the Ascii Character: 1001101
Ex: Calculate Hamming Code for the message 1001101
At the sender side:

1	2	3	4	5	6	7	8	9	10	11
R1	R2	D1	R4	D2	D3	R8	D4	D5	D6	D7

1	2	3	4	5	6	7	8	9	10	11
		1		0	0	1		1	0	1

Calculating the r values :

r1:

1	3	5	7	9	11
	1	0	1	1	1

r1=0 (since there is even parity in 1's)

r2:

2	3	6	7	10	11
	1	0	1	0	1

r2=1 (since there is no even parity in 1's)

r4:

4	5	6	7
	0	0	1

r4=1 (since there is no even parity in 1's)

r8:

8	9	10	11
	1	0	1

r8=0 (since there is even parity in 1's)

so now the transmitted frame is :

1	2	3	4	5	6	7	8	9	10	11
0	1	1	1	0	0	1	0	1	0	1

At the receiver side:

Assume that the above transmitted data is received by and 7th bit has been changed from 1 to 0. The receiver takes the transmission and recalculates four new VRCs using the same sets of bits used by the sender plus the relevant parity (r) for each set. Then it assembles the new parity values into a binary number in order of r position (r1,r2,r4,r8).

1	2	3	4	5	6	7	8	9	10	11
0	1	1	1	0	0	0	0	1	0	1

r1:

1	3	5	7	9	11
0	1	0	0	1	1

r1=0 (since there is no even parity in 1's)

r2:

2	3	6	7	10	11
1	1	0	0	0	1

r2=1 (since there is no even parity in 1's)

r4:

4	5	6	7
1	0	0	0

r4=1 (since there is no even parity in 1's)

r8:

8	9	10	11
0	1	0	1

r8=0 (since there is even parity in 1's)

so, now the result is r8r4r2r1 is 0111 means that the error at the 7th position.

The receiver takes the transmission and recalculates four new VRCs using the same sets of bits used by the sender plus the relevant parity(r) bit for each set. Then it assembles the new parity values into a binary number in order of r position (r8, r4, r2, r1). It gives the binary number 0111(7 in decimal), which is precise location of the bit in error.
Once the bit is identified the receiver can reverse its value and correct the error.

Flow Control: The feature of data link layer is Flow Control. It is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgement from the receiver. The flow of data must not beat the receiver's capability. Any receiving device has a limited speed at which it can process incoming data and a limited amount of memory to store incoming data. So the receiving device must be able to inform the sender before reaching to these limits and request the source to send less no of frames or to stop transmission for some time from now onwards. The rate of such processing is always slower than the rate of transmission. That is why each receiver maintains a buffer to reserve for storing incoming data until they processed. Two methods have been developed to control the flow of data communication. They are Elementary Data Link Control Protocols and Sliding Window Protocols.

Elementary Data Link Control Protocols:

1. An unrestricted Simplex Protocol: Using this protocol data is transmitted in one direction only. Both the transmitting and receiving devices are always ready to transmit the data. Processing time is ignored. Infinite buffer space is available. Since only one person is using the channel, frame never damage or lost by intermediate devices. That is the communication channel is assumed to be error free and the receiver is assumed to be able to process all the input infinitely quickly. This is strictly an unrealistic protocol. It is called with a nick name "utopia".

Sender-site algorithm for the simplest protocol

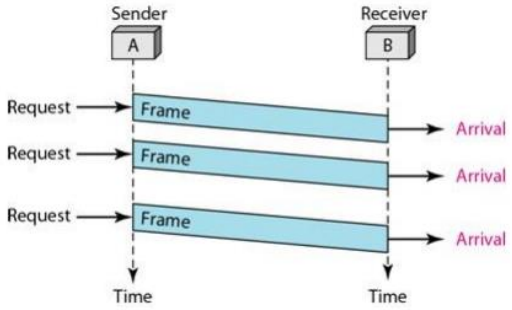
```
1  while(true)                    // Repeat forever
2  {
3    WaitForEvent();              // Sleep until an event occurs
4    if(Event(RequestToSend))     //There is a packet to send
5    {
6       GetData();
7       MakeFrame();
8       SendFrame();              //Send the frame
9    }
10 }
```

Receiver-site algorithm for the simplest protocol

```
1  while(true)                    // Repeat forever
2  {
3    WaitForEvent();              // Sleep until an event occurs
4    if(Event(ArrivalNotification)) //Data frame arrived
5    {
6       ReceiveFrame();
7       ExtractData();
8       DeliverData();            //Deliver data to network layer
9    }
10 }
```

The simplest protocol is a unidirectional protocol in which data frames are traveling in only one direction from the sender to receiver. We assume that the receiver can immediately handle any frame it receives with a negligible processing time. The data link layer of the receiver immediately removes the header from the frame and hands the data packet to its network layer. The data link layer at the sender side gets data from its

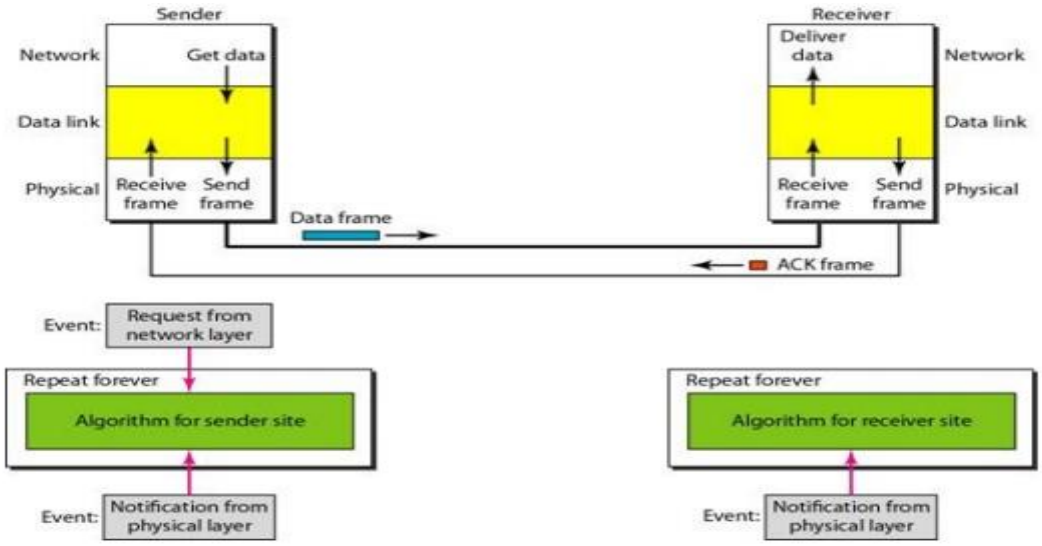
network layer, makes a frame and sends it. The data link layer at the receiver side receives a frame from its physical layer, extracts data from the frame, and delivers the data to its network layer.



2. Simplex Stop and Wait Protocol: Using this protocol we can achieve half duplex communication. Protocols in which the sender sends one frame and waits for an acknowledgement before proceeding to next frame are known as Stop and Wait protocols. Although the data traffic is simplex here, data goes only from the sender to receiver, frames do travel in both the directions. So the channel must be capable of handling bidirectional information transfer. This protocol follows:

1. First the sender sends a frame, then the receiver sends an ack frame.
2. next the sender sends another frame and the receiver sends another ack frame and so on.

Therefore a half duplex channel will work here. This protocol is also assumed to be error free.



Sender Algorithm:

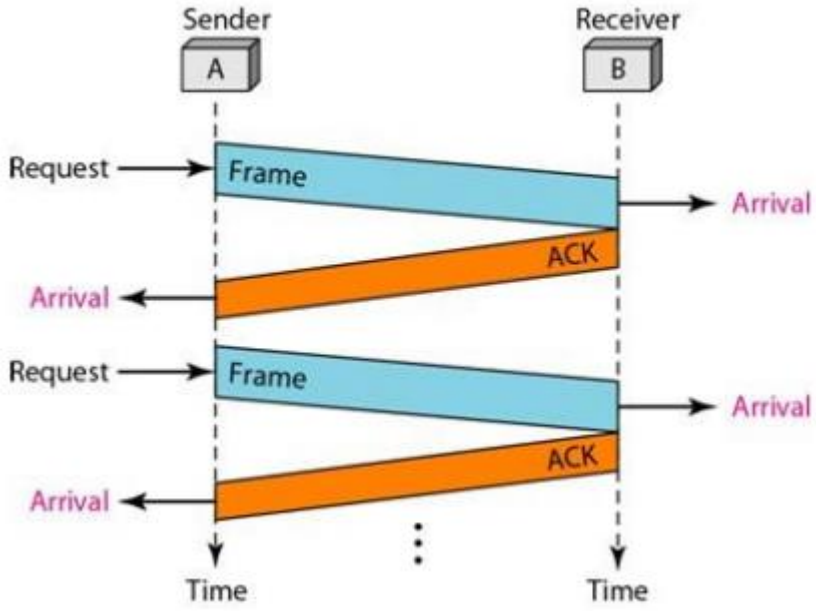
```
1  while(true)                          //Repeat forever
2  canSend = true                       //Allow the first frame to go
3  {
4    WaitForEvent();                    // Sleep until an event occurs
5    if(Event(RequestToSend) AND canSend)
6    {
7       GetData();
8       MakeFrame();
9       SendFrame();                    //Send the data frame
10      canSend = false;                //Cannot send until ACK arrives
11   }
12   WaitForEvent();                    // Sleep until an event occurs
13   if(Event(ArrivalNotification)      // An ACK has arrived
14    {
15      ReceiveFrame();                 //Receive the ACK frame
16      canSend = true;
17    }
18 }
```

Receiver Algorithm:

```
1  while(true)                          //Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrives
5    {
6       ReceiveFrame();
7       ExtractData();
8       Deliver(data);                  //Deliver data to network layer
9       SendFrame();                    //Send an ACK frame
10   }
11 }
```

Here two events can occur, a request from the network layer or an arrival notification from the physical layer. After a frame is sent, the algorithm must ignore another network layer request until that frame is acknowledged. We know that two arrival events cannot happen one after another because the channel is error-free and doesn't duplicate the frames. The requests from the network layer, however, may happen one after another without an arrival event in between. We need to prevent the immediate sending of the data frame. We have used a method canSend variable that can either be true or false. When a frame is sent, the variable is set to false to indicate that a new network request cannot be sent until canSend is true. When an ACK is received, canSend is set to true to allow the sending of the next frame. After the data frame arrives, the receiver sends an ACK frame to acknowledge and allow the sender to send the next frame.

Ex:

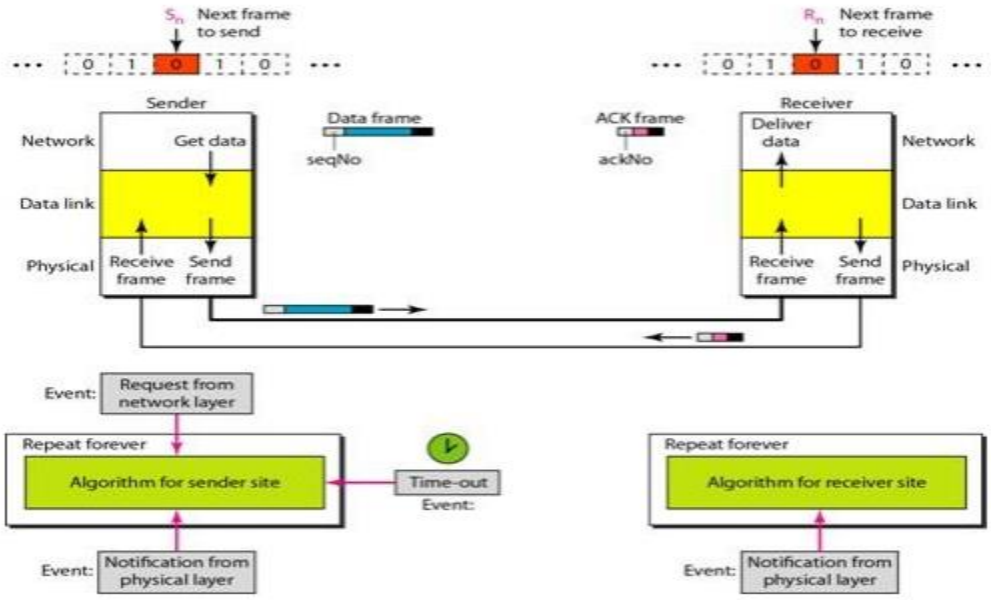


3. Simplex Protocol for a Noisy Channel (Stop and Wait ARQ): Here we consider a normal situation of a communication channel that makes errors. Frames may be either damaged or lost completely. However, if we assume that if a frame is damaged during transmission the receiver detects this while computing the checksum. If a damaged frame arrives at the destination, it would be discarded. The sender waits for an ack till the time out period expires. Then the sender sends the frame again. This process will be repeated until the frame arrives safely.

The Stop & Wait Automatic repeat request adds a simple error control mechanism to the Stop & Wait Protocol. To detect and correct corrupted frames, we need to add redundancy bits to our data frame. When the frame arrives at the receiver side, it is checked and if it is corrupted, it is silently discarded. Lost frames are more difficult to handle than corrupted frames. In our previous protocols, there was no way to identify a frame. The received frame could be the correct one, or a duplicate, or a frame out of order. The solution is to number the frames. When the receiver receives a data frame that is out of order, this means that frames were either lost or duplicated. The lost frames need to be resent in this protocol.

The sender keeps a copy of the sent frame and starts a timer while sending a frame. If the timer expires and there is no ACK for the sent frame, the frame is resent, the copy is held, and the timer is restarted. Since an ACK frame can also be corrupted and lost, it too needs redundancy bits and a sequence number. The ACK frame for this protocol has a sequence number field. The figure below shows the design of the Stop & Wait ARQ protocol. The sending device keeps a copy of the last frame transmitted until it receives an acknowledgement for that frame. A data frame uses a seqNo, an ACK frame uses an

ackNo. 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.



Sender Algorithm:

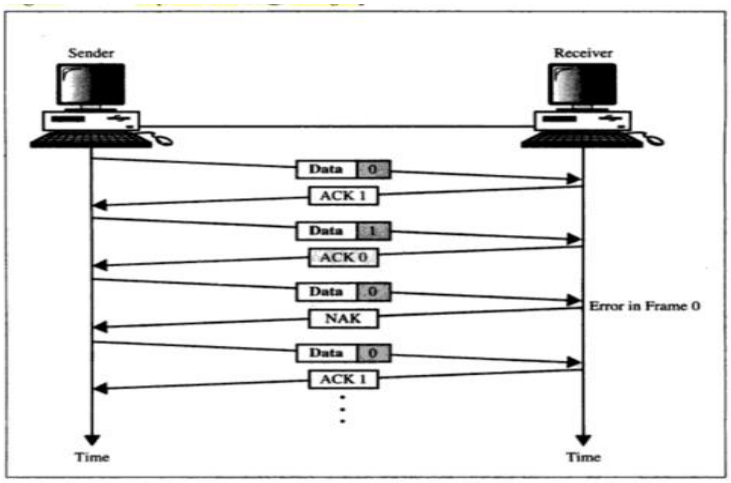
```
1  Sn = 0;                              // Frame 0 should be sent first
2  canSend = true;                      // Allow the first request to go
3  while(true)                          // Repeat forever
4  {
5    WaitForEvent();                    // Sleep until an event occurs
6    if(Event(RequestToSend) AND canSend)
7    {
8       GetData();
9       MakeFrame(Sn);                  //The seqNo is Sn
10      StoreFrame(Sn);                 //Keep copy
11      SendFrame(Sn);
12      StartTimer();
13      Sn = Sn + 1;
14      canSend = false;
15   }
16   WaitForEvent();                    // Sleep
17     if(Event(ArrivalNotification)    // An ACK has arrived
18     {
19       ReceiveFrame(ackNo);           //Receive the ACK frame
20       if(not corrupted AND ackNo == Sn) //Valid ACK
21         {
22           Stoptimer();
23           PurgeFrame(Sn-1);          //Copy is not needed
24           canSend = true;
25         }
26      }
27
28     if(Event(TimeOut)                // The timer expired
29     {
30      StartTimer();
31      ResendFrame(Sn-1);              //Resend a copy check
32     }
33 }
```

Receiver Algorithm:

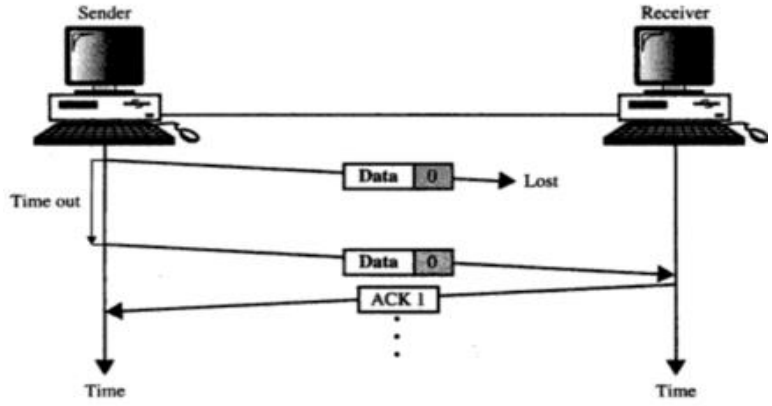
```
1  Rn = 0;                          // Frame 0 expected to arrive first
2  while(true)
3  {
4    WaitForEvent();                // Sleep until an event occurs
5    if(Event(ArrivalNotification))  //Data frame arrives
6    {
7       ReceiveFrame();
8       if(corrupted(frame));
9          sleep();
10      if(seqNo == Rn)              //Valid data frame
11      {
12       ExtractData();
13        DeliverData();             //Deliver data
14        Rn = Rn + 1;
15      }
16       SendFrame(Rn);             //Send an ACK
17   }
18 }
```

Error Controlling: To handle these situations, we are taking the above protocol with timer mechanism. Then it becomes to handle noisy channel. So now we discus about error handling mechanisms. Protocols in which sender waits for a +ve ack before transmitting next frame is called +ve Ack with retransmission or AR!(Automatic Repeat and Request).

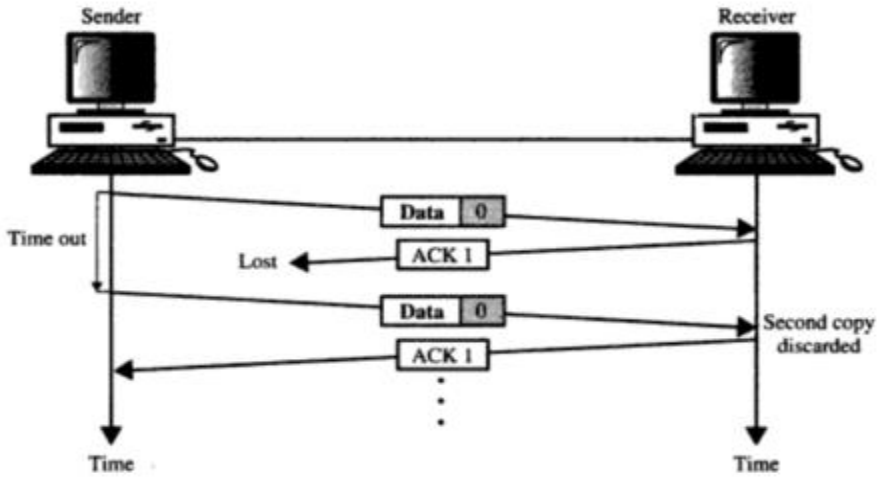
Damaged Frames: when a frame is received by the receiver which contains error, it returns a negative ack frame and the sender retransmits the damaged frame. For example, in the following diagram the sender sends a frame 0, the receiver returns an ack1, by specifying frame 0 is arrived safely and I am now expecting frame 1. So the sender transmits the next frame.



Lost Data Frames: as we discussed earlier, the sender will have a timer. Timer starts every time a data frame is transmitted by the sender. If the frame never reaches it to the receiver, the receiver can never acknowledge it positiveky or negatively. The sending device waits for an ack or nak frame until its timer time outs. Then it retransmits the lost frame and restarts the timer and waits for an acknowledgement.



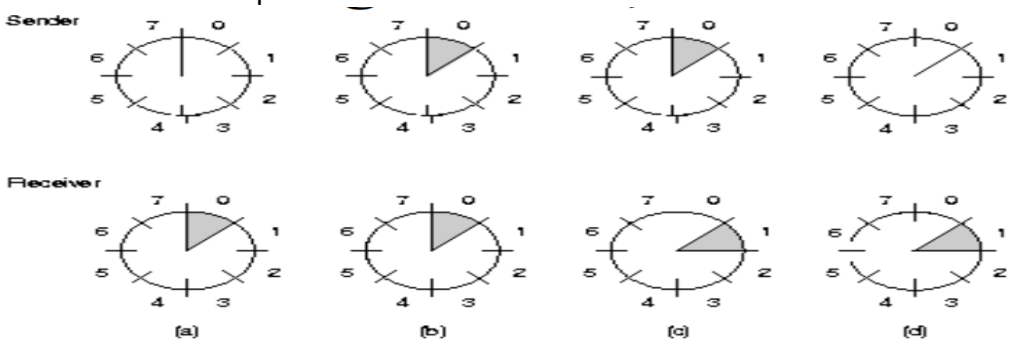
Lost Acknowledgements: Here in this case, the frames are reached to the receiver. Receiver found either to accept the frame, or to reject the frame based on the checksum provided or calculated b. by specifying this acceptance or reject receiver transfers ACK or NAK frames. These ACK frames are lost during the transmission. The source waits until its timer expires and retransmits the data frame. The receiver checks the sequential number of new data frame. If the lost one is NAK frame then the receiver accepts a new frame and sends an ACK to this new frame. If the lost one is an ACK, the receiver recognizes this as a duplicate packet and acknowledges its receipt and then discards it and waits for the next frame.



Sliding Window Protocols:

In all the previous protocols data frames were transmitted in one direction only. But in some practical situations there is a need to distribute data in both directions. One way of achieving full duplex transmission is to have two separate channels and use each one for simplex data traffic in different directions. One is known as a forward channel to distribute data frames and the other is a reverse channel for acknowledgements. In both the cases the bandwidth of the reverse channel is almost entirely wasted. In effect, the user is paying for both the channels and using the capacity of only one channel.

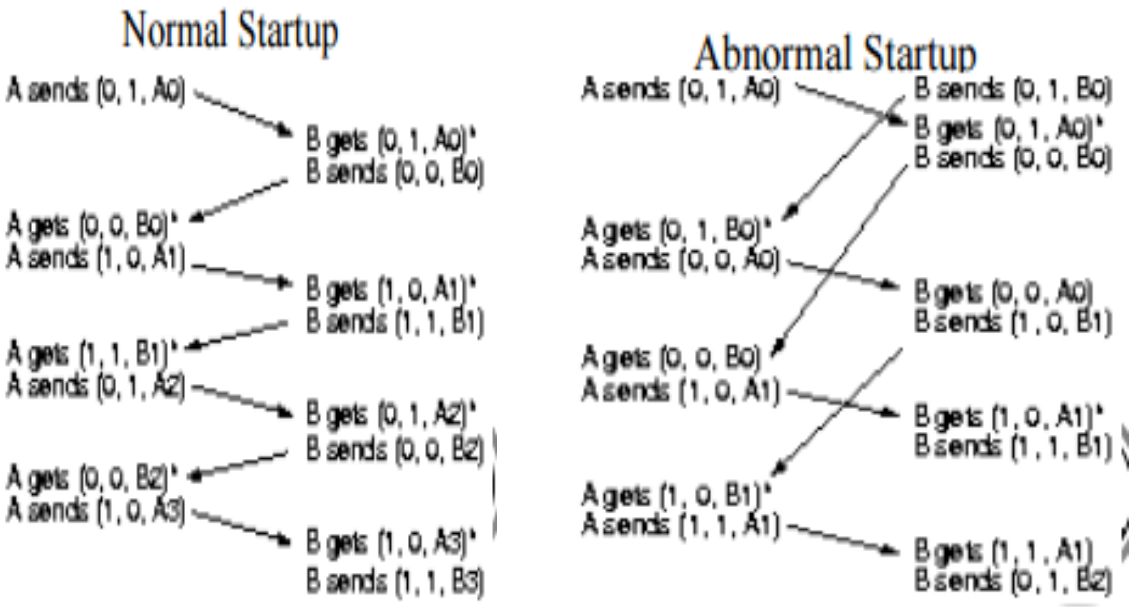
Although interleaving data and control frames on the same channel is an improvement over having two separate physical circuits. When a data frame arrives, instead of sending acknowledgements separately, the receiver restarts and waits for the next outgoing packet. The technique of temporarily delaying the outgoing acknowledgements, so that they can be hooked (inserted) onto the next outgoing packet is known as **piggybacking.** The main advantage of using piggybacking over having separate ack frames is a better use of available channel bandwidth. The next three protocols are more robust and continue to function even under pathological conditions. All three belongs to a class of protocols called **sliding window protocols.** The three differ among themselves in terms of efficiency, complexity, and buffer requirements. The following is an example of transmission using these protocols with a sliding window of size 1and with a 3 bit sequence number.



a) **Intitially b) after the frame has been sent c) after the first frame has been received d) after the first ack has been received.**

One-bit Sliding Window Protocol: This is a protocols which follows sliding window size of 1 with stop and wait mechanism. Sender transmits a frame and waits for its acknowledgement before sending the next one. The acknowledgement field contains the number of the last frame received without error. However a peculiar situation happens if both sides simultaneously send an initial packet. This synchronization difficulty is shown in the following diagram. In the first case we are dealing with normal operation, in the second case peculiar condition is discussed. In the first case, B waits for A's first frame before sending its data. By following this method every frame is is accepted. Each frame arrival brings a new packet for the network layer without creating any duplicate packets. But where as in the second case both the users start communicating at once. The result is their first frames itself collide each other and are

discarded. Half of the frames in communication re duplicates even though there are no transmission errors.

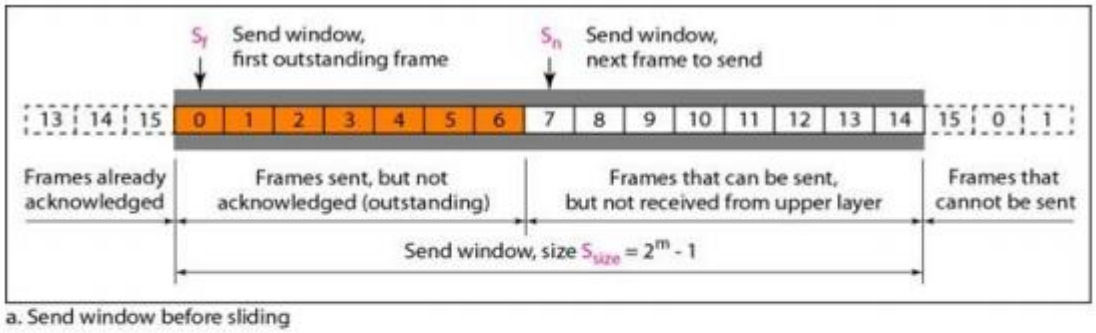


Sliding window protocol using Go back N (Go Back N ARQ): While transmitting data from one end to another end if any frame is damaged, the receiver simply discards all the following frames without sending any acknowledgements to these discarded frames. If the sender did not receive any acknowledgement corresponding to these frames with in the time out period, sender will retransmit all the frames starting with the damaged frame. This approach can waste a lot of bandwidth if the error rate is high. This strategy corresponds to a window size of 1. If one frame is lost or damaged, all frames sent since the last frame acknowledged are retransmitted.

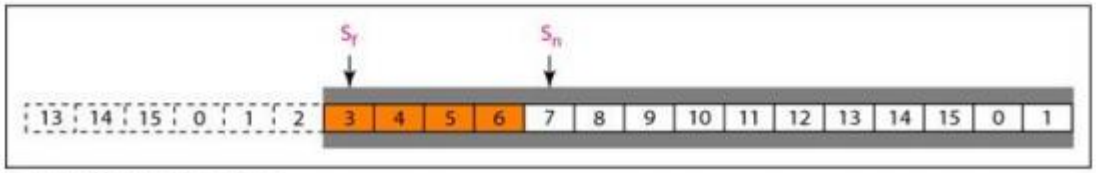
In this protocol, the sliding window defines the range of packets that the sender and receiver can send. The sender's capability is shown using the send sliding window and the receiver capability is shown in the receiver sliding window. The window at any time divides the possible sequence numbers into four regions.

1. The first region, from the left of the window, defines the sequence numbers belonging to frames that are already acknowledged. The sender doesn't worry about these frames and keeps no copies of them.
2. The second region defines the range of sequence numbers belonging to the frames that are sent and have an unknown status. The sender needs to wait to find out if these frames have been received or were lost. We call these outstanding frames.
3. The third range defines the range of sequence numbers for frames that can be sent, however, the corresponding data packets have not yet been received from the network layer.

4. Finally, the fourth region defines sequence numbers that cannot be used until the window slides.

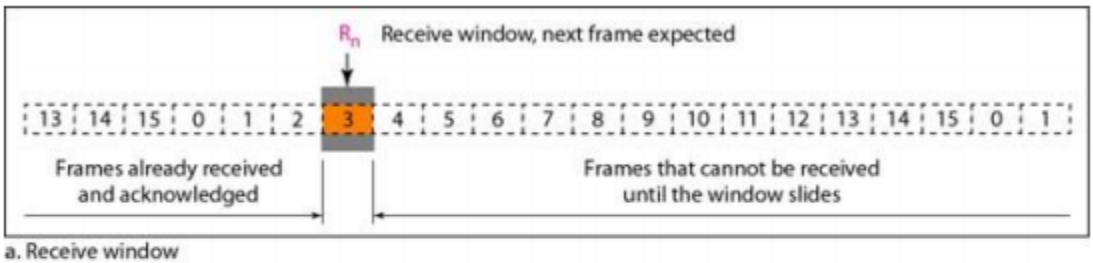


a. Send window before sliding

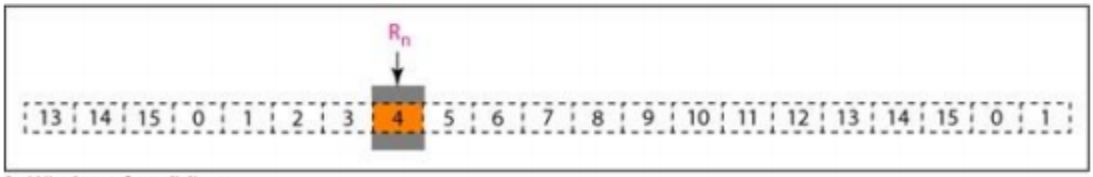


b. Send window after sliding

The receive window makes sure that the correct data frames are received and that the correct acknowledgements are sent. The size of the receiver window is always 1. The receiver is always looking for the arrival of a specific frame. Any frame arriving out of order is discarded and needs to be resent



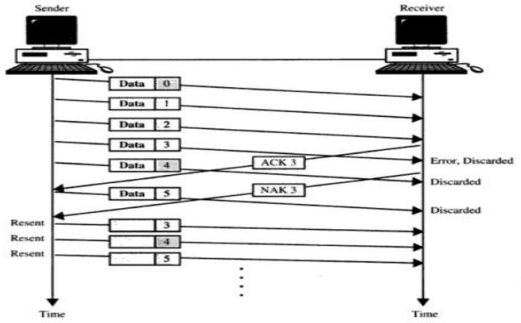
a. Receive window



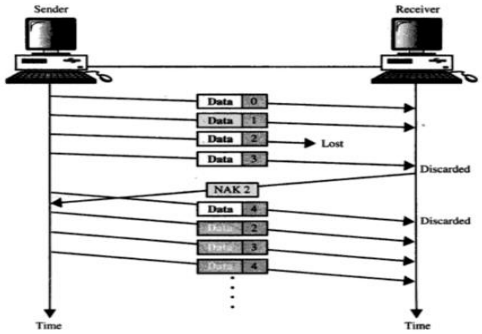
b. Window after sliding

Damaged Frames: Imagine what shall we do if frames 0,1,2 and 3 have been transmitted, but the first ack received is a NAK 3?. Here the term NAK corresponds tp a +ve ack by specifying all frames received prior to the damaged frame. And the other meaning is data frame 3 is corrupted and should be resent. For example if the sender transmitted 6 frames, while receiving the data suppose of the receiver found frame 3 is corrupted. In this case an ack 3 has been returned, telling the sender that frames 0,1,2 have been accepted. Why because all these frames arrived safely before data 3. Since packet 3 found in corrupted position a NAK 3 is sent immediately and frames 4,5 are

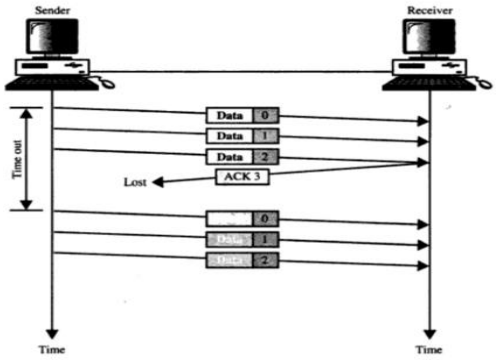
discarded as they come without bothering either they arrived safely or unsafely. So the sender retransmits all the 3 frames (3,4,5).



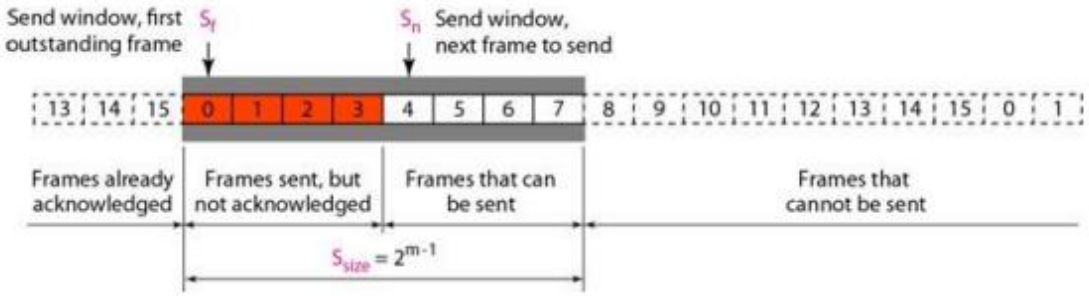
Lost Frames: Sliding window protocols require that data frames be transmitted in sequential order. Due to noise if one or more frames are lost by the intermediate routers during the transmission, the next following frames will not be in sequential order at the receiver side. The receiver checks seq no of each packet and discovers that one or more frames are not present and sends an NAK for the first missed frame. After receiving this NAK sender has to transfer all the subsequent frames starting from the first missed frame. This scenario is shown in the following diagram.



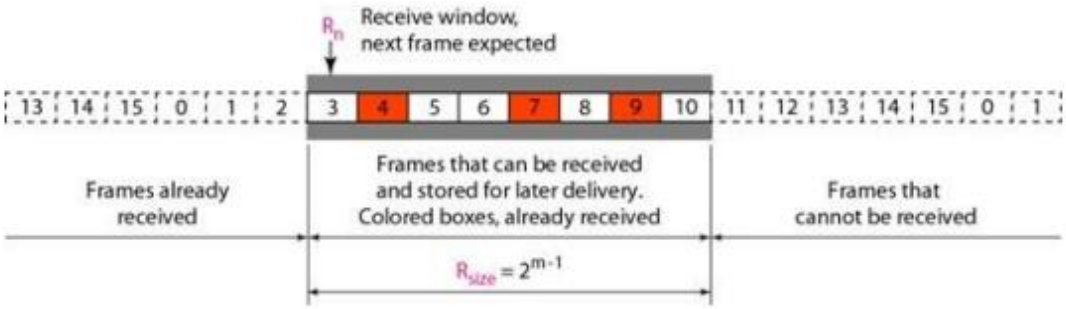
Lost Acknowledgements: If the ACK sent by the receiver is lost, the sender will not wait a long time. To recover from this the sender maintains a timer when it reaches to the maximum time out period, if it didn't receive any ACK within this interval, it simply retransmits all packets starting from missed ACK. This scenarios is shown in the following diagram.



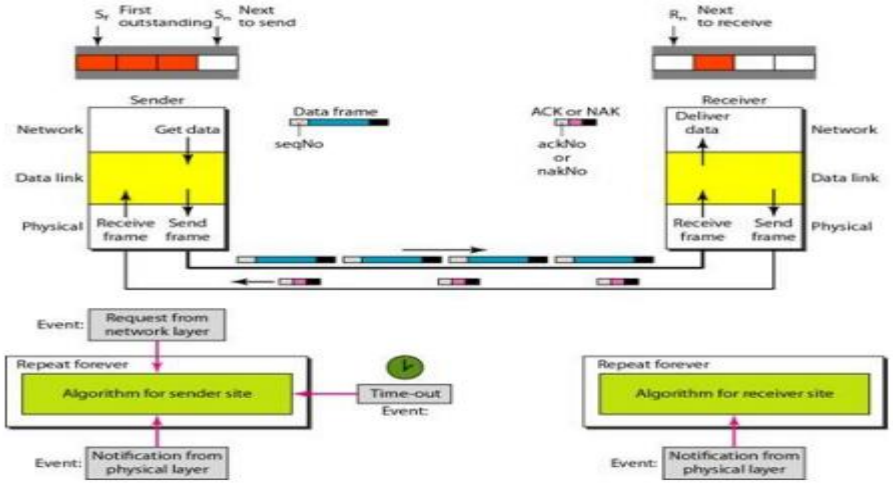
Sliding with Selective Repeat or Reject (Selective Repeat ARQ): In this protocol only a specific damaged or corrupted, lost data frame is retransmitted instead of retransmitting all the following frames. All these subsequent (following) frames are buffered by the data link layer. As soon as it receives the damaged frame safely, the receiver sorts all the frames in order and insert the retransmitted frame in a proper place.



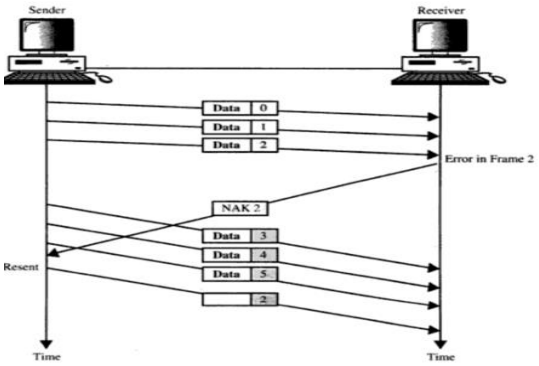
Receiver Window



The Selective Repeat Protocol allow as many frames as the size of the receive window to arrive out of order and be kept until there is a set of in order frames to be delivered to the network layer. Because the sizes of the send window and receive window are the same, all the frames in the send frame can arrive out of order and be stored until they can be delivered. The design in this case is to some extent like Go Back N ARQ protocol



Damaged Frames: As shown in the following diagram, frames 0 and 1 are received but not yet acknowledged. Packet 2 arrives and is found errored frame. So a NAK frame is sent for frame 2 and it buffers all the following frames. Sender will retransmit the second frame. When the receiver accepts this retransmitted one they put the frame in proper place.



Lost Frames: Although the Frames can be accepted out of sequence, they cannot be acknowledged out of sequence. If a frame is lost, the next frame will arrive out of sequence. The receiver sends a NAK corresponding to this frame. After receiving this frame, receiver orders all the frames and sends ack for all the following frames. If any one of them is corrupted it will ask the sender to retransmit that packet.

Lost Acknowledgement: Lost ACK and NAK frames are treated by selective reject just like go back n. if no ack arrives in the allotted time, sender retransmits all the frames which have not been acknowledged. In most cases, receiver reorganizes duplicate packets and discards these duplicate packets.

Data Link Protocols: A data link protocol is a set of specifications used to implement data link layer. Data link protocols can be divided into two subgroup: asynchronous protocols and synchronous protocols. Asynchronous protocols treat each character in a bit stream independently. Synchronous protocols take the whole bit stream and chop it into characters of equal size.

Synchronous Protocols: The speed of synchronous transmission makes it he better choice, over asynchronous transmission. Protocols governing synchronous transmission can be divided into two classes: character oriented protocols and bit oriented protocols.
Character oriented protocols: These are also called byte oriented protocols. They interpret a transmission frame or packet as a succession of characters, each usually composed of one byte (eight bits). All control information is in the form of an existing character encoding system.

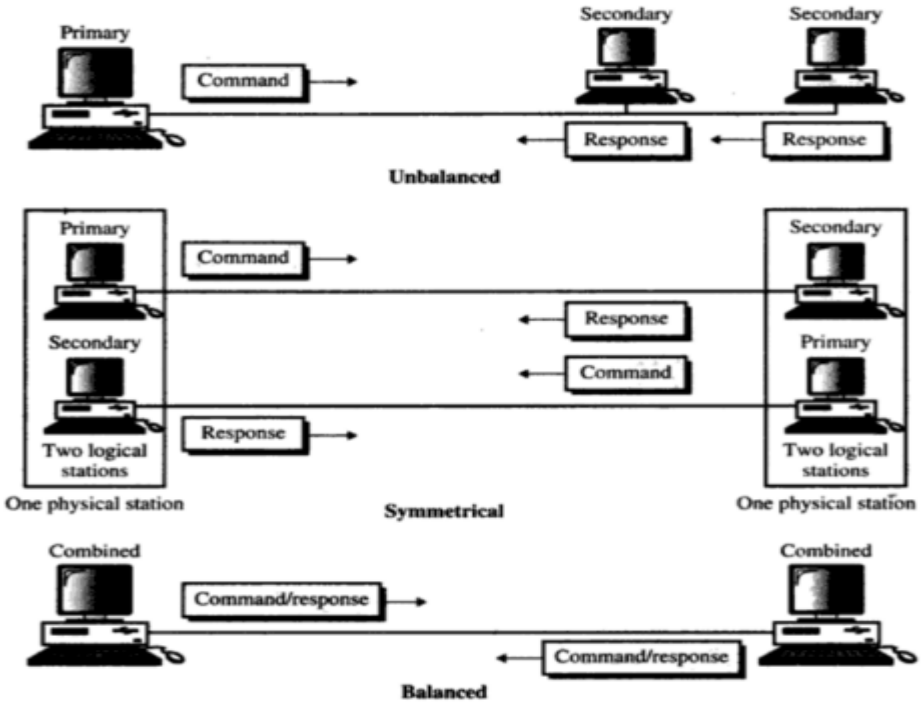
Bit oriented protocols: Interpret a transmission frame or packet as a succession of individual bits, make meaningful by either placement in the frame. Control information in a bit oriented protocol can be one or multiple bits depending on the information embodied in the pattern.

HDLC: In 1975, IBM pioneered the development of bit oriented protocols with synchronous data link control (SDLC) and lobbied the ISO to make SDLC as a standard. In 1979, ISO answered with HDLC which was based on SDLC. HDLC is a bit oriented data link protocol designed to support both half duplex and full duplex communications over point to point and multi point networks. There are 3 different types of stations: primary, secondary and combined.

The relationship of hardware devices on a link is configuration. Primary, secondary and combined stations can be configured in three ways: unbalanced, symmetrical, and balanced. Any of these configurations can support both half duplex and full duplex transmission.

An unbalanced configuration is one in which one device is primary and the others are secondary. Unbalanced configurations can be point-to-point if only two devices are involved; more often they are multipoint, with one primary controlling several conditions.

A symmetrical configuration is one in which each physical station on a link consists of two logical stations, one a primary and the other secondary. Separate lines link the primary aspect of one physical station to the secondary aspect of another physical station. A symmetrical configuration behaves like an unbalanced configuration except that control of the link can shift between the two stations.



A balanced configuration is one in which both stations in a point to point topology are of the combined type. The stations are linked by a single line that can be controlled by either station.

Modes of Communication: A mode in HDLC is a relationship between two devices involved in an exchange; the mode describes who controls the link. HDLC supports three modes of communication between stations: normal response mode (NRM), asynchronous response mode (ARM), and asynchronous balanced mode(ABM).
NRM: It is a standard primary-secondary relationship. In this mode a secondary device must have permission from the primary device before transmitting. Once permission has been granted, the secondary may initiate a response transmission of one or more frames containing data.
ARM: A Secondary may initiate a transmission without permission from the primary whenever the channel is idle. ARM does not alter the primary-secondary relationship in any other way. All transmissions from a secondary must still be made to the primary for relay to a final destination.
ABM: All station are equal and therefore only combined stations connect in point to point are used. Either combined station may initiate transmission with the other combined station without permission.

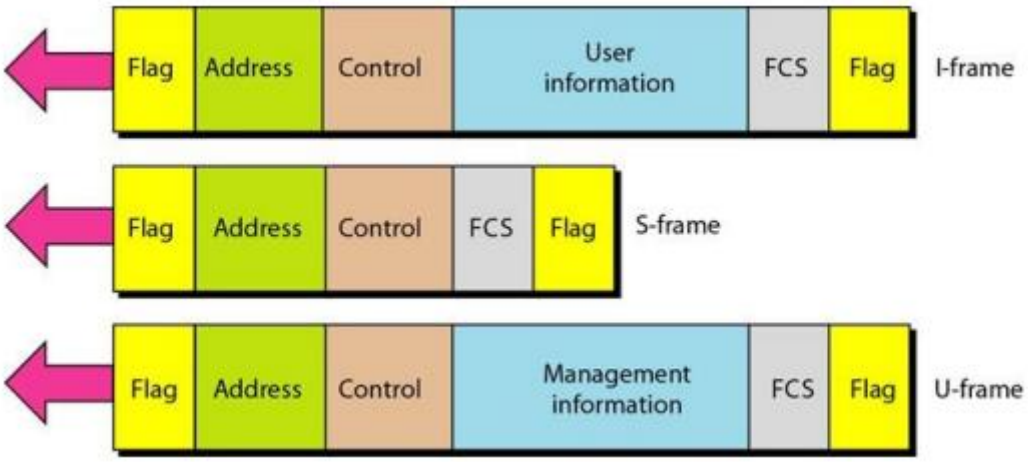
Frames: HDLC defines three types of frames: **information frames (I-franes), supervisory frames(S-frames), and unnumbered frames (U-frames).** Each type of frame works as an envelope for the transmission of a different type of message.

I-frames are used to transport the user data and control information relating to user data. The first bit defines the type of frame. If the first bit of the control field is 0, this means the frame is an I– frame. The next 3 bits, called N(s), define the sequence number of the frame. The last 3 bits, called N(R), represent acknowledgement number of last transmitted frame. A bit between N(S) and N(R) represents Poll or Final. If there are some continuous frames after this frame, it is represented as '0' which denotes poll. If this frame is the last frame of transmission, then it is represented as '1' which denotes final.

S-frames are used only to transport control information, primarily data link layer flow and error controls. S – frames do not have information fields. If the first 2 bits of the control field is 10, this means the frame is S – frame. The last 3 bits, called N(R), corresponds to the acknowledgement number (ACK) or negative acknowledgement number (NAK) depending on the type of S – frame. The 2 bits called code is used to define the type of S – frame itself. With 2 bits, we can have four types of S – frames, as follows
1. Receive Ready (RR): represented by '00'
2. Receive Not Ready (RNR): represented by '10'
3. Reject (REJ): represented by '01'
4. Selective Reject (SREJ): represented by '11'

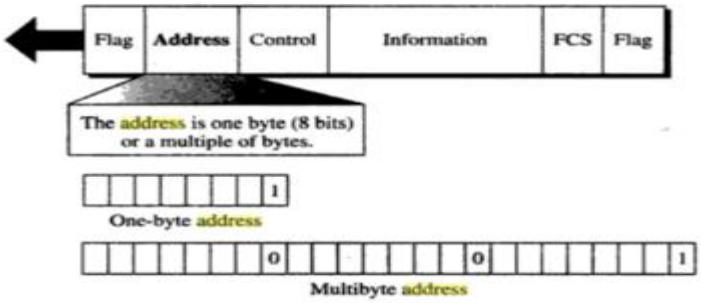
U-frames are reserved for system management. Information carried by U-frames is intended for managing the link itself. U–frames are used to exchange session management and control information between connected devices. Unlike S – frames, U

– frames contain an information field, but one used for system management information, not user data. As with S – frames, however, much of the information carried by U – frames are contained in codes included in the control filed. U – frame codes are divided into two sections: a 2- bit prefix before P/F bit and a 3-bit suffix after the P/F bit. Together, these two segments

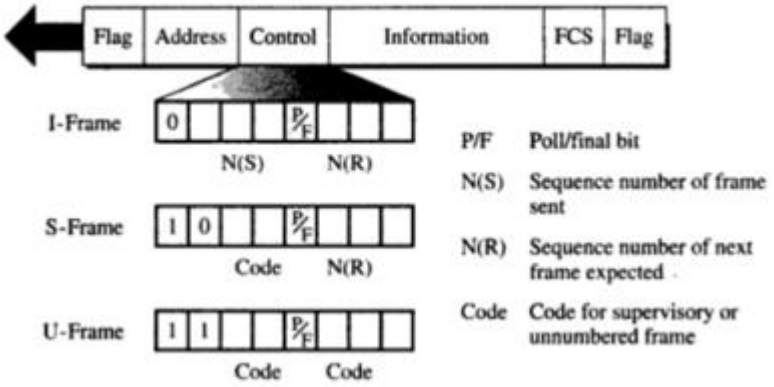


Flag: It is an 8-bit sequence with a bit stream 01111110 that identifies both the beginning and end of a frame and serves as a synchronization pattern for the receiver.

Address: Address of the secondary station that is either the originator or destination of the frame. If a primary station creates a frame, it contains a to address. If a secondary creates the frame, it contains a from 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 terminals. Larger networks require multiple byte address fields. If the address field is only one byte, the least significant bit is always 1. If the address is more than one byte, all bytes but the last one will end with 0; only the last will end with 1. Ending each intermediate byte with 0 indicates to the receiver that there are more address bytes to come.



Control field: It is a one or two byte segment of the frame used for flow management. Control fields differ depending on frame type. If the first bit of the control field is 0, the frame is an I-frame. If the first two bites 10. It is an S-frame. If both the first and seconds bits are 1 then it is a U-frame. The control fields of all three types of frames contain a bit called the poll/final (P/f) bit.



Information Field: Contains the user's data in an I-frame, and network management information in a u-frame. Its length can vary from one network to another but is always fixed in a network. An s-frame has no information field.

FCS: Frame check sequence is an error detection field. It contains either a 2 byte or 4 byte CRC.

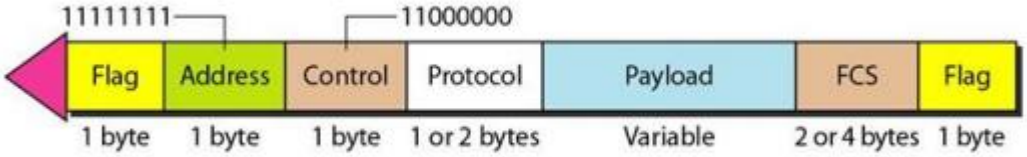
Point to Point Protocol:

HDLC is a protocol that can be used for both point – to – point and multipoint configurations. The most common protocol for point – to – point access is the Point – to – Point Protocol (PPP). Unlike HDLC, PPP is a byte oriented protocol. Point – to – Point

PPP Services:

1. PPP defines the format of the frames to be exchanged between devices.
2. PPP defines how two devices can negotiate the establishment of the link and the exchange of data.
3. PPP defines how network layer data are encapsulated in the data link frame.
4. PPP defines how two devices can authenticate each other.
5. PPP provides connections over multiple links

Frame Format:



Flag: This field is of length one byte which starts and ends with 01111110.

Address: This is of length one byte which is a constant and is set to 11111111 (broadcast address). During negotiation, this byte can be omitted by the two parties.

Control: This field is set to a constant value 11000000. PPP does not provide flow control. Error control is also limited to error detection. It means that this field is not required at all. Means both the sender and receiver can agree, during negotiation to omit this byte.

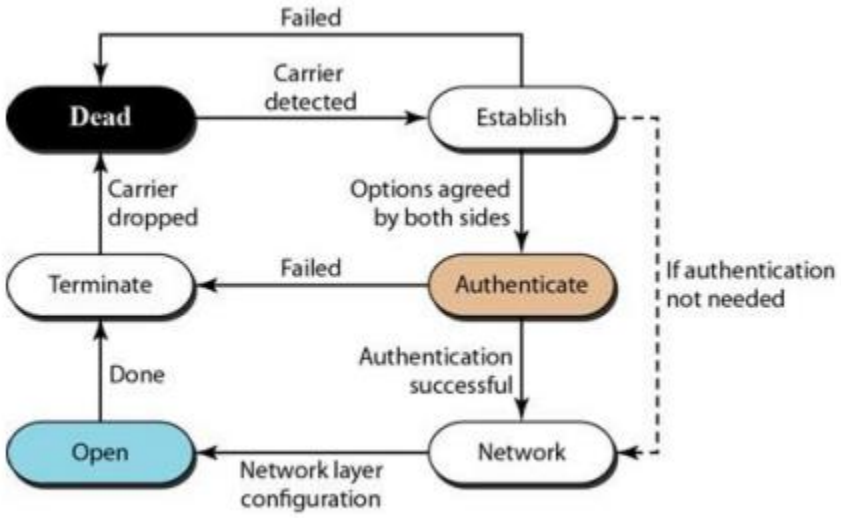
Protocol: It defines what is being carried in the data field: either user data or other information. This field is of length two bytes, but the two parties can agree to use only one byte.

Payload: This is of variable length which carries either user data or other information with a default maximum length of 1500 bytes; but this can be changed during negotiation.

FCS: Frame Check sequence is of length 2 or 4 bytes and uses either CRC16 or CRC32.

Byte Stuffing: PPP is a byte oriented protocol, the flag in PPP is a byte and needs to be escaped whenever it appears in a data section of the frame. The escape byte is 01111101, which means that every time if the flag appears in data, this extra byte is stuffed to inform the receiver that the next byte is not a flag.

Transition Phases:



Dead: At this stage, link is not being used. There is no active carrier at the physical layer and the line is quiet.

Establish: When any one of the node starts communicating with other, this phase will start. In this phase many options are going to be negotiated between the two parties.
If the negotiation is successful the system goes to the authentication phase or directly to the networking phase.

Authenticate: This is optional phase. The two parties may decide either to go in this phase or to skip this.

Network: In the network phase, negotiation for the network layer protocols takes place. PPP supports multiple protocols at the network layer.

Open: In this phase, data transmission takes place.

Terminate: In this phase, connection is terminated.

Multiplexing: PPP is a data link layer protocol but it uses another set of protocols to establish a link, authenticate the parties involved, and carry the network layer data. Three set of protocols are defined to make PPP powerful. They are:

1. one Link Control Protocol : Responsible for establishing, maintaining, configuring, and terminating links. It also provides negotiation to set options between the end parties.

2. Two Authentication Protocols: Plays very important role in PPP because PPP is designed for use over dial-up links where verification of user identity is necessary. They are of two types.
i) Password Authentication Protocol
ii) Challenge Handshake Authentication Protocol

3. Several Network Control Protocols: PPP carries various network layer protocols data (Internet, OSI, Xerox, DECnet, AppleTalk, Novel and so on).

Multi-Link PPP: PPP was originally designed for a single channel Point to Point physical link. In multi-link PPP a logical PPP frame is divided into several actual PPP frames. A segment of the logical frame is carried in the payload of an actual PPP frame.