Unit-3

The Data Link Layer

Content:

1. The data link layer issues
2. Error detection and correction
3. Elementary data link protocols
4. Sliding window protocols
5. Examples of data link protocol: HDLC

I. Introduction-

In this chapter we will study the design principles for layer 2, the data link layer. This study deals with the algorithms for achieving reliable, efficient communication between two adjacent machines at the data link layer. By adjacent, we mean that the two machines are connected by a communication channel that acts conceptually like a wire (e.g., a coaxial cable, telephone line, or point-to-point wireless channel). The essential property of a channel that makes it "wire-like" is that the bits are delivered in exactly the same order in which they are sent.

After an introduction to the key design issues present in the data link layer, we will start our study of its protocols by looking at the nature of errors, their causes, and how they can be detected and corrected. Then we will study a series of increasingly complex protocols, each one solving more and more of the problems present in this layer. Finally, we will conclude with an examination of protocol modeling and correctness and give some examples of data link protocols.

3.1 Data Link Layer Design Issues:

The data link layer has a number of specific functions it can carry out. These functions include:

1. Providing a well-defined service interface to the network layer.

2. Dealing with transmission errors.

3. Regulating the flow of data so that slow receivers are not swamped by fast senders.

To accomplish these goals, the data link layer takes the packets it gets from the network layer and encapsulates them into frames for transmission. Each frame contains a frame header, a payload field for holding the packet, and a frame trailer, as illustrated in Fig. 3-1.

Frame management forms the heart of what the data link layer does. In the following sections we will examine all the above-mentioned issues in detail.

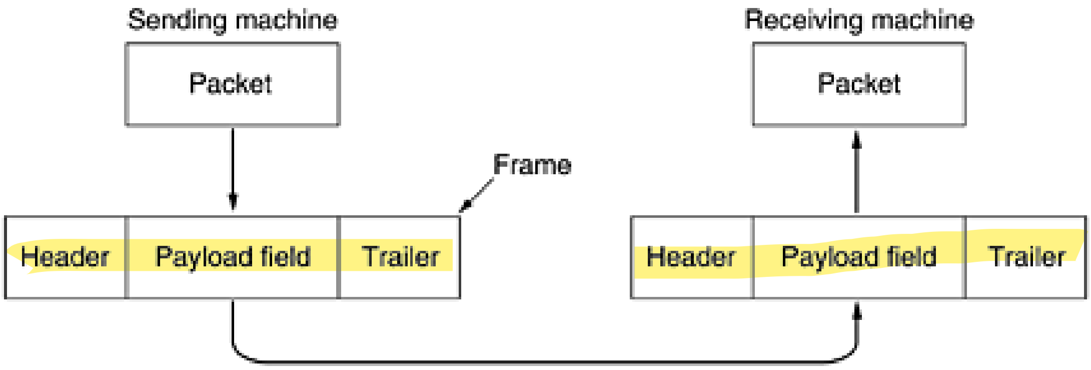


Figure 3-1. Relationship between packets and frames.

3.1.1 Services Provided to the Network Layer:

The function of the data link layer is to provide services to the network layer.

The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine.

On the source machine is an entity, call it a process, in the network layer that hands some bits to the data link layer for transmission to the destination.

The job of the data link layer is to transmit the bits to the destination machine so they can be handed over to the network layer there, as shown in Fig. 3-2(a).

The actual transmission follows the path of Fig. 3-2(b), but it is easier to think in terms of two data link layer processes communicating using a data link protocol. For this reason, we will implicitly use the model of Fig. 3-2(a) throughout this chapter.

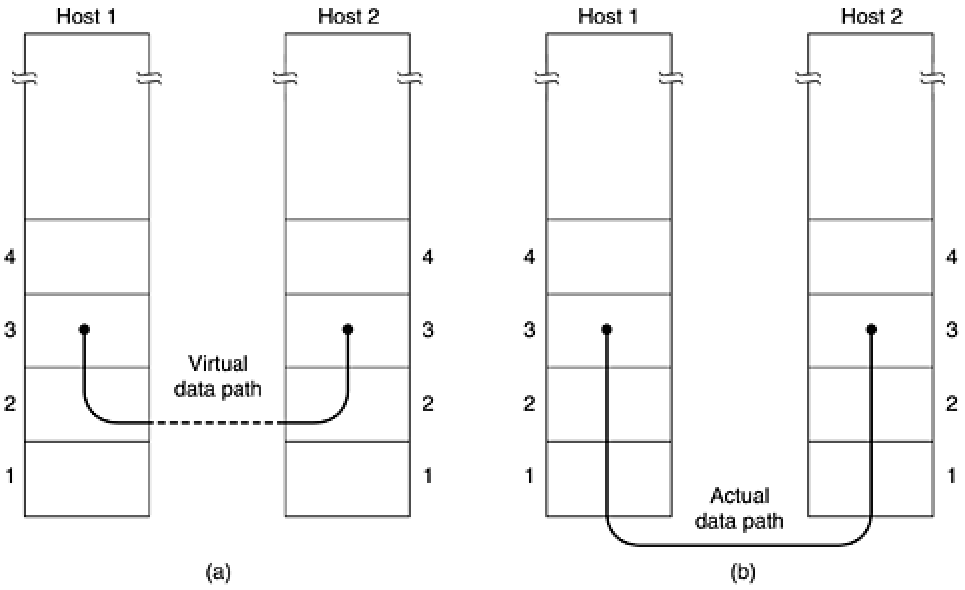


Figure 3-2. (a) Virtual communication. (b) Actual communication.

The data link layer can be designed to offer various services. The actual services offered can vary from system to system. Three reasonable possibilities that are commonly provided are

1. Unacknowledged connectionless service.
2. Acknowledged connectionless service.
3. Acknowledged connection-oriented service.

- **Unacknowledged connectionless service:**

Unacknowledged connectionless service consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them.

No logical connection is established beforehand or released afterward.

If a frame is lost due to noise on the line, no attempt is made to detect the loss or recover from it in the data link layer.

This class of service is appropriate when the error rate is very low so that recovery is left to higher layers.

- **Acknowledged connectionless service:**

When this service is offered, there are still no logical connections used, but each frame sent is individually acknowledged.

In this way, the sender knows whether a frame has arrived correctly.

If it has not arrived within a specified time interval, it can be sent again. This service is useful over unreliable channels, such as wireless systems.

It is perhaps worth emphasizing that providing acknowledgements in the data link layer is just an optimization, never a requirement.

The network layer can always send a packet and wait for it to be acknowledged.

If the acknowledgement is not forthcoming before the timer expires, the sender can just send the entire message again.

If individual frames are acknowledged and retransmitted, entire packets get through much faster.

On reliable channels, such as fiber, the overhead of a heavyweight data link protocol may be unnecessary, but on wireless channels, with their inherent unreliability, it is well worth the cost.

- **Acknowledged connection-oriented service:**

The most sophisticated service the data link layer can provide to the network layer is connection-oriented service.

With this service, the source and destination machines establish a connection before any data are transferred.

Each frame sent over the connection is numbered, and the data link layer guarantees that each frame sent is indeed received.

Furthermore, it guarantees that each frame is received exactly once and that all frames are received in the right order.

With connectionless service, in contrast, it is conceivable that a lost acknowledgement causes a packet to be sent several times and thus received several times.

Connection-oriented service, in contrast, provides the network layer processes with the equivalent of a reliable bit stream.

When connection-oriented service is used, transfers go through three distinct phases.

In the first phase, the connection is established by having both sides initialize variables and counters needed to keep track of which frames have been received and which ones have not.

In the second phase, one or more frames are actually transmitted. In the third and final phase, the connection is released, freeing up the variables, buffers, and other resources used to maintain the connection.

3.1.2 Framing:

To provide service to the network layer, the data link layer must use the service provided to it by the physical layer. What the physical layer does is accept a raw bit stream and attempt to 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, and they may have different values. It is up to the data link layer to detect and, if necessary, correct errors.

The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame.

When a frame arrives at the destination, the checksum is recomputed. If the newly-computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it (e.g., discarding the bad frame and possibly also sending back an error report).

Breaking the bit stream up into frames is more difficult than it at first appears. One way to achieve this framing is to insert time gaps between frames, much like the spaces between words in ordinary text.

However, networks rarely make any guarantees about timing, so it is possible these gaps might be squeezed out or other gaps might be inserted during transmission.

Since it is too risky to count on timing to mark the start and end of each frame, other methods have been devised. In this section we will look at four methods:

1. Character count.

2. Flag bytes with byte stuffing.

3. Starting and ending flags, with bit stuffing.

4. Physical layer coding violations.

The first framing 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 follow and hence where the end of the frame is.

This technique is shown in Fig. 3-4(a) for four frames of sizes 5, 5, 8, and 8 characters, respectively.

The trouble with this algorithm is that the count can be garbled by a transmission error.

For example, if the character count of 5 in the second frame of Fig. 3-4(b) becomes a 7, the destination will get out of synchronization and will be unable to locate the start of the next frame. Even if the checksum is incorrect so the destination knows that the frame is bad, it still has no way of telling where the next frame starts.

Sending a frame back to the source asking for a retransmission does not help either, since the destination does not know how many characters to skip over to get to the start of the retransmission.

For this reason, the character count method is rarely used anymore.

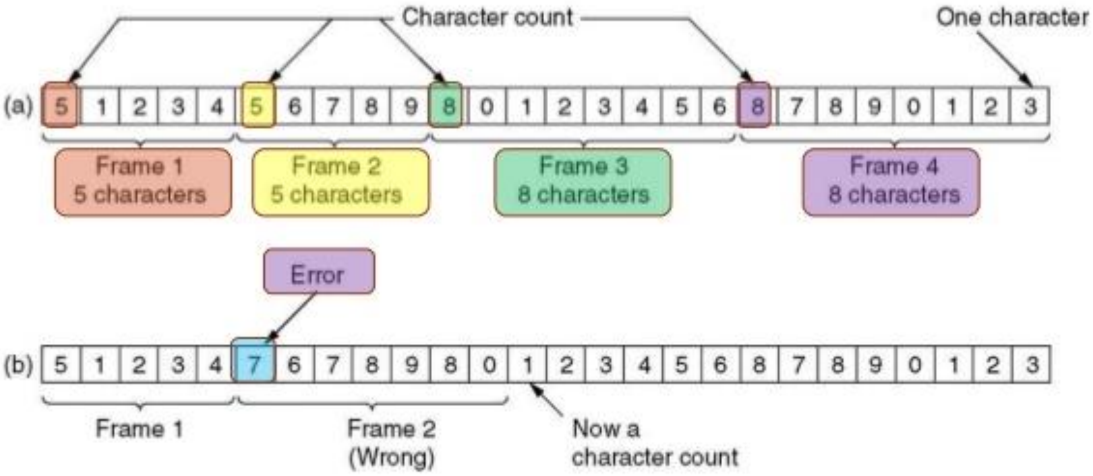


Figure 3-4. A character stream. (a) Without errors. (b) With one error.

The second framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes.

In the past, the starting and ending bytes were different, but in recent years most protocols have used the same byte, called a flag byte, as both the starting and ending delimiter, as shown in Fig. 3-5(a) as FLAG.

In this way, if the receiver ever loses synchronization, it can just search for the flag byte to find the end of the current frame. Two consecutive flag bytes indicate the end of one frame and start of the next one.

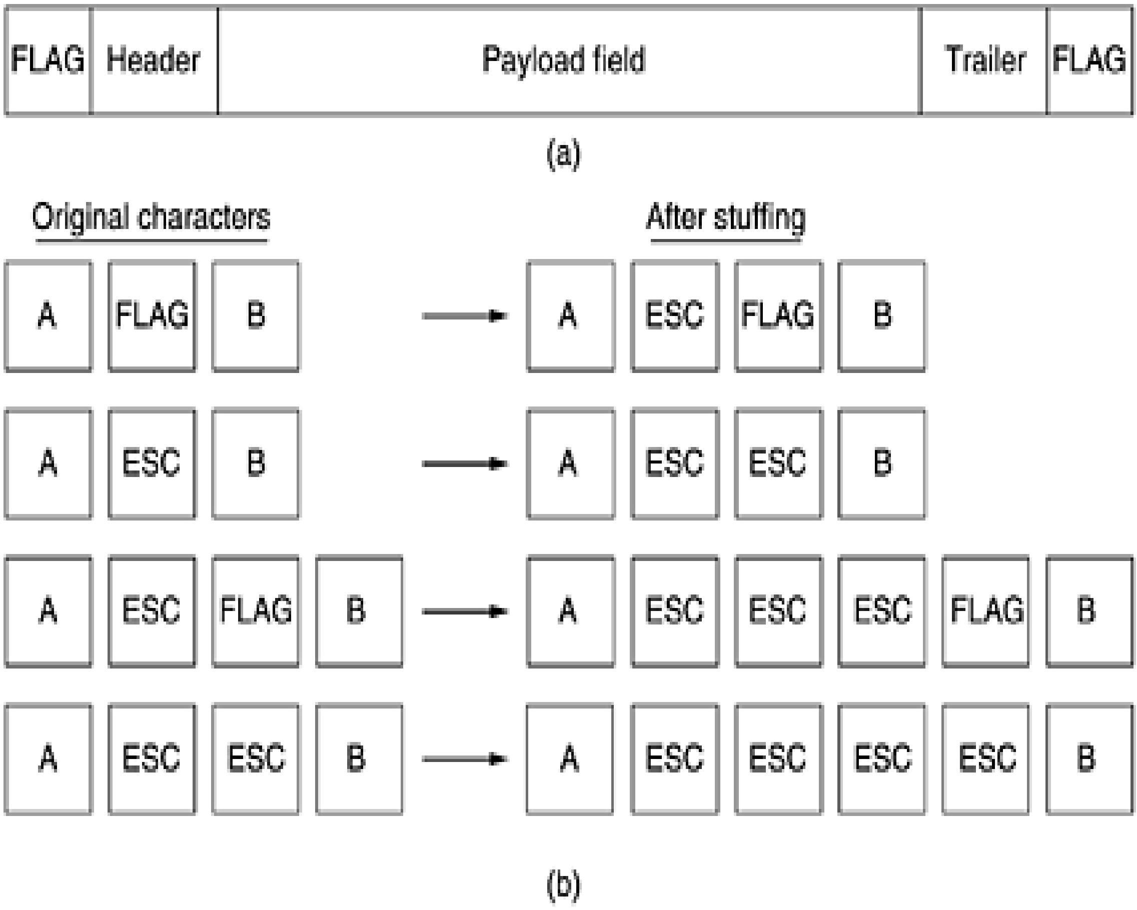


Figure 3-5. (a) A frame delimited by flag bytes. (b) Four examples of byte sequences before and after byte stuffing.

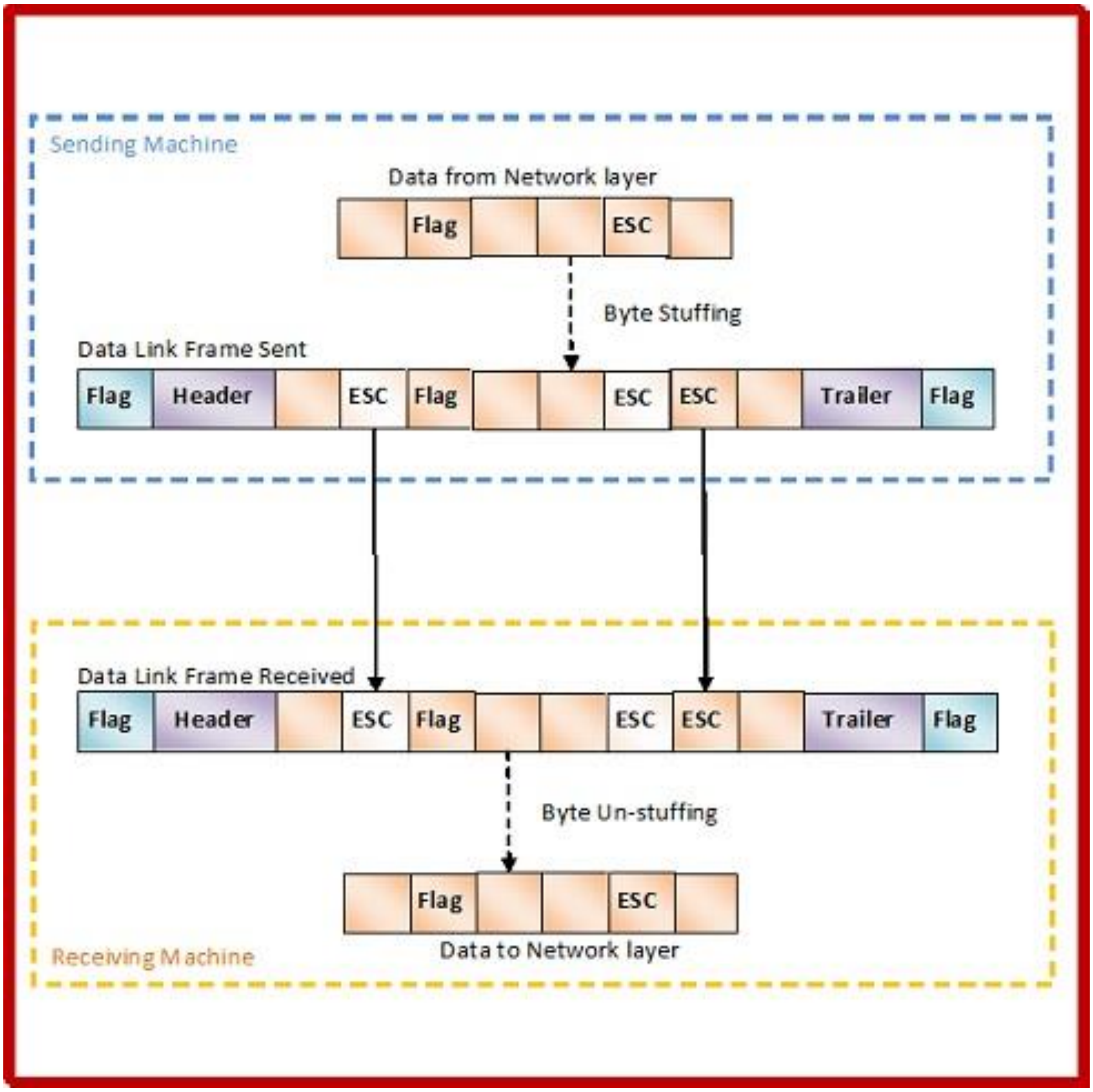


Figure 3-5. (c) Byte stuffing mechanism

A serious problem occurs with this method when binary data, such as object programs or floating-point numbers, are being transmitted. It may easily happen that the flag byte's bit pattern occurs in the data. This situation will usually interfere with the framing.

One way to solve this problem is to have the sender's data link layer insert a special escape byte (ESC) just before each ''accidental'' flag byte in the data. The data link layer on the receiving end removes the escape byte before the data are given to the network layer.

This technique is called byte stuffing or character stuffing. Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it.

Of course, the next question is: What happens if an escape byte occurs in the middle of the data? The answer is that it, too, is stuffed with an escape byte.

Thus, any single escape byte is part of an escape sequence, whereas a doubled one indicates that a single escape occurred naturally in the data. Some examples are shown in Fig. 3-5(b).

In all cases, the byte sequence delivered after de-stuffing is exactly the same as the original byte sequence.

The new technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character.

It works like this. Each frame begins and ends with a special bit pattern, 01111110 (in fact, a flag byte).

Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream.

This bit stuffing is analogous to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically de-stuffs (i.e., deletes) the 0 bit.

Just as byte stuffing is completely transparent to the network layer in both computers, so is bit stuffing.

If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110.

Figure 3.6 (a)Explains Bit Stuffing Mechanism (b) gives an example of bit stuffing.

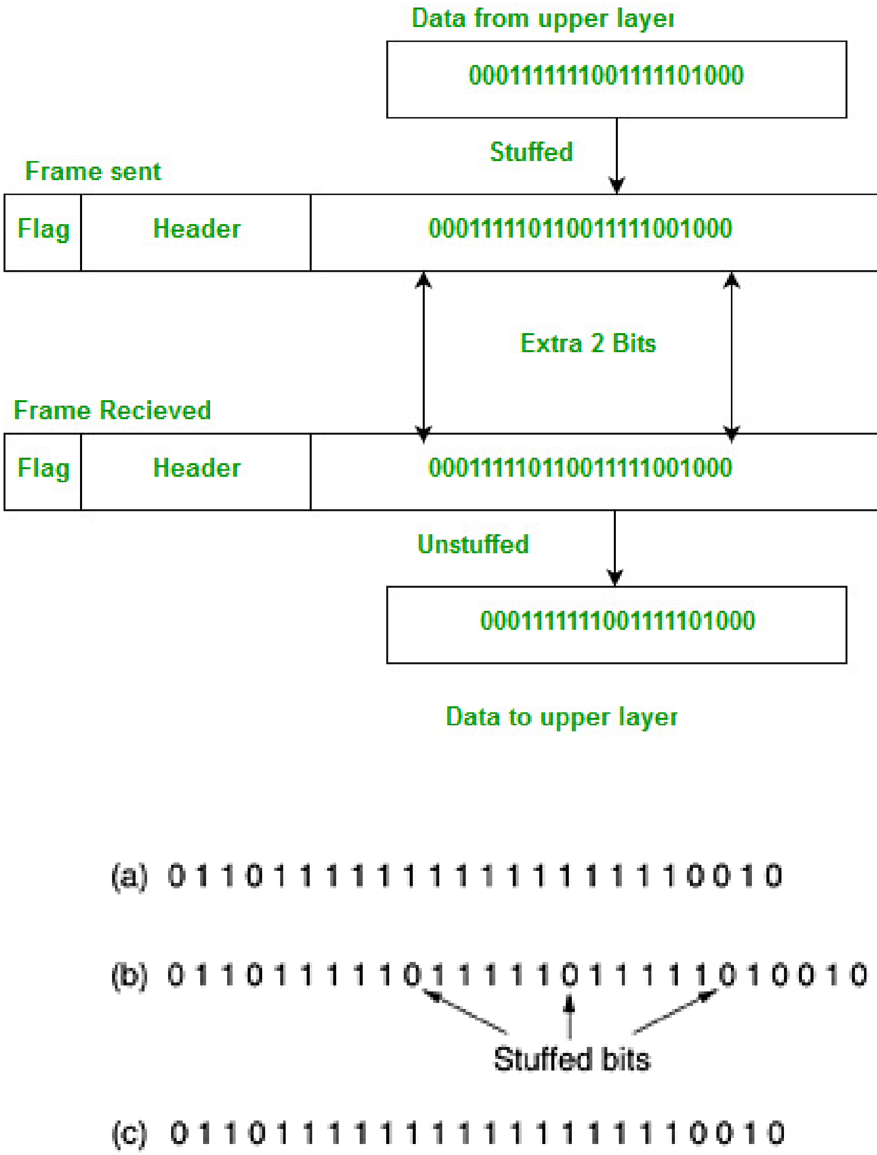


Figure 3-6. (a) Bit Stuffing Mechanism (b) An example of Bit Stuffing.

With bit stuffing, the boundary between two frames can be unambiguously recognized by the flag pattern.

Thus, if the receiver loses track of where it is, all it has to do is scan the input for flag sequences, since they can only occur at frame boundaries and never within the data.

The last method of framing is only applicable to networks in which the encoding on the physical medium contains some redundancy.

As a final note on framing, many data link protocols use a combination of a character count with one of the other methods for extra safety.

When a frame arrives, the count field is used to locate the end of the frame.

Only if the appropriate delimiter is present at that position and the checksum is correct is the frame accepted as valid. Otherwise, the input stream is scanned for the next delimiter.

3.1.3 Error Control

Error control in data link layer is the process of detecting and correcting data frames that have been corrupted or lost during transmission.

The usual way to ensure reliable delivery is to provide the sender with some feedback about what is happening at the other end of the line.

Typically, the protocol calls for the receiver to send back special control frames bearing positive or negative acknowledgements about the incoming frames.

If the sender receives a positive acknowledgement about a frame, it knows the frame has arrived safely.

On the other hand, a negative acknowledgement means that something has gone wrong, and the frame must be transmitted again.

In case of lost or corrupted frames, the receiver does not receive the correct data-frame and sender is ignorant about the loss.

Data link layer follows a technique to detect transit errors and take necessary actions, which is retransmission of frames whenever error is detected or frame is lost. The process is called Automatic Repeat Request (ARQ).

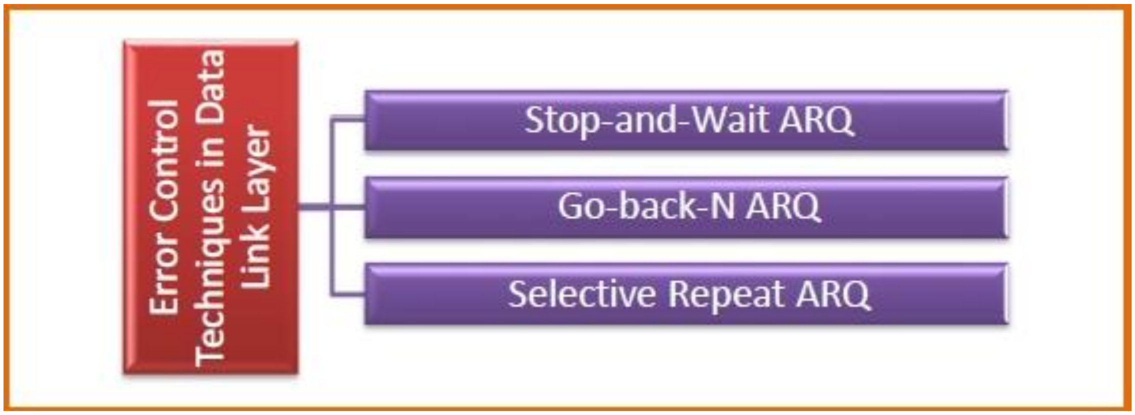
Phases in Error Control

The error control mechanism in data link layer involves the following phases –

- **Detection of Error** – Transmission error, if any, is detected by either the sender or the receiver.
- **Acknowledgment** – acknowledgment may be positive or negative.
    - **Positive ACK** – On receiving a correct frame, the receiver sends a positive acknowledge.
    - **Negative ACK** – On receiving a damaged frame or a duplicate frame, the receiver sends a negative acknowledgment back to the sender.
- **Retransmission** – The sender maintains a clock and sets a timeout period. If an acknowledgment of a data-frame previously transmitted does not arrive before the timeout, or a negative acknowledgment is received, the sender retransmits the frame.

Error Control Techniques

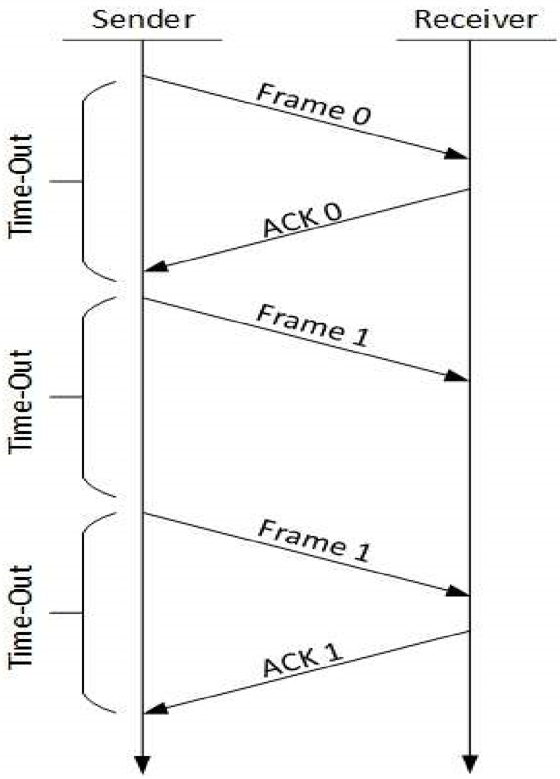
There are three main techniques for error control –



- **Stop and Wait ARQ**

This protocol involves the following transitions −

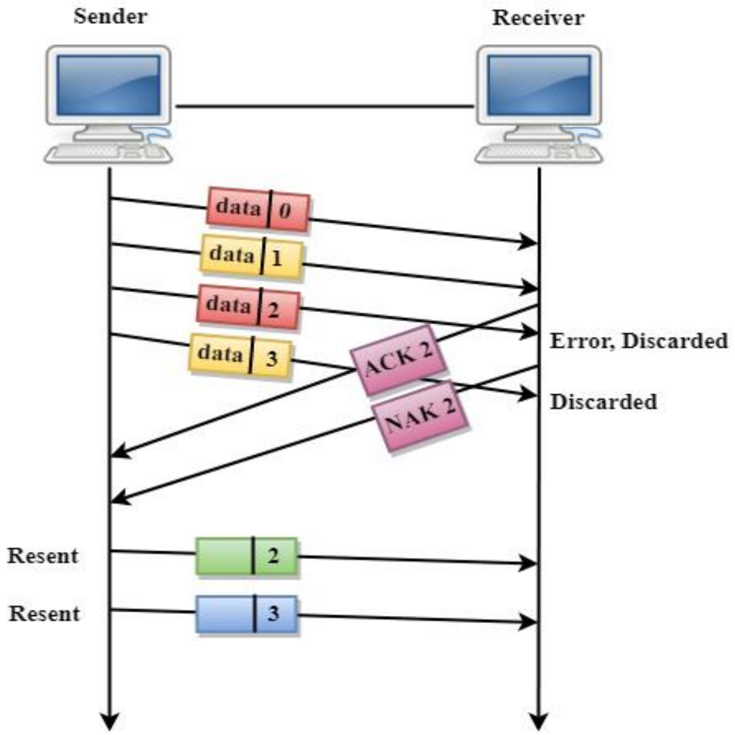
- A timeout counter is maintained by the sender, which is started when a frame is sent.
- If the sender receives acknowledgment of the sent frame within time, the sender is confirmed about successful delivery of the frame. It then transmits the next frame in queue.
- If the sender does not receive the acknowledgment within time, the sender assumes that either the frame or its acknowledgment is lost in transit. It then retransmits the frame.
- If the sender receives a negative acknowledgment, the sender retransmits the frame.



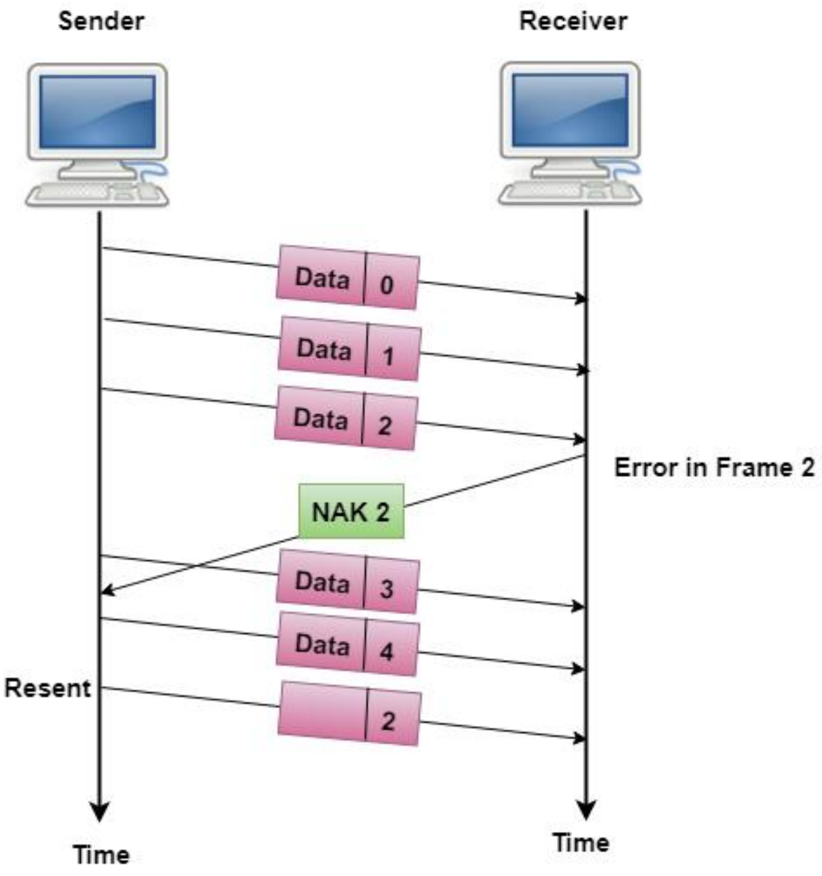
- **Go-Back-N ARQ**

The working principle of this protocol is –

- The sender has buffers called sending window.The sender sends multiple frames based upon the sending-window size, without receiving the acknowledgment of the previous ones.
- The receiver receives frames one by one. It keeps track of incoming frame's sequence number and sends the corresponding acknowledgment frames.
- After the sender has sent all the frames in window, it checks up to what sequence number it has received positive acknowledgment.
- If the sender has received positive acknowledgment for all the frames, it sends next set of frames.
- If sender receives NACK or has not received any ACK for a particular frame, it retransmits all the frames after which it does not receive any positive ACK.



- **Selective Repeat ARQ**
  - Both the sender and the receiver have buffers called sending window and receiving window respectively.
  - The sender sends multiple frames based upon the sending-window size, without receiving the acknowledgment of the previous ones.
  - The receiver also receives multiple frames within the receiving window size.
  - The receiver keeps track of incoming frame's sequence numbers, buffers the frames in memory.
  - It sends ACK for all successfully received frames and sends NACK for only frames which are missing or damaged.
  - The sender in this case, sends only packet for which NACK is received.



3.1.4 Flow Control:

Another important design issue that occurs in the data link layer (and higher layers as well) is what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them.

This situation can easily occur when the sender is running on a fast (or lightly loaded) computer and the receiver is running on a slow (or heavily loaded) machine.

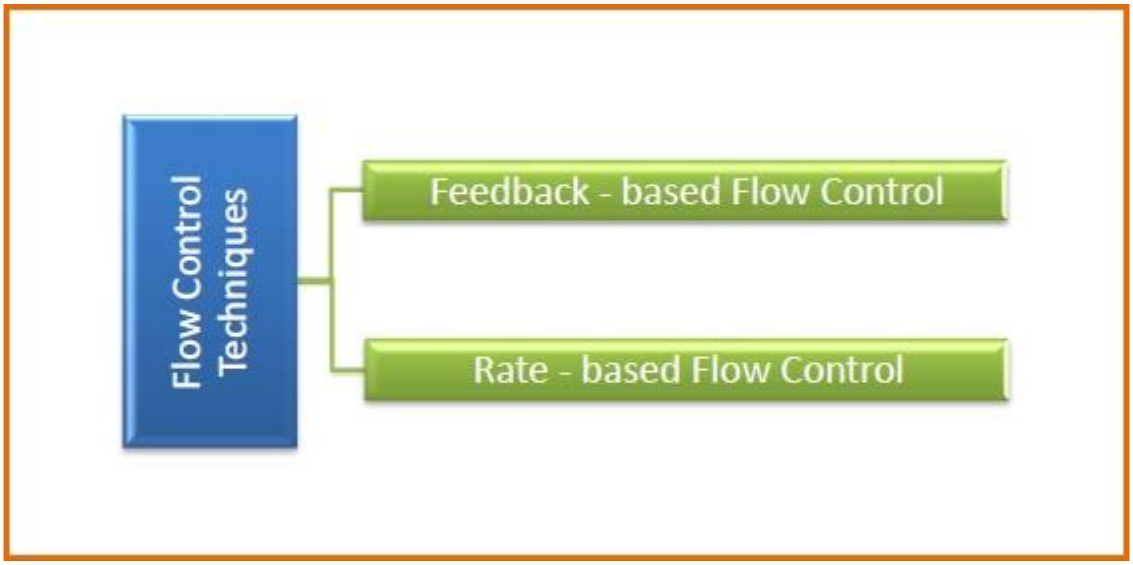
The sender keeps pumping the frames out at a high rate until the receiver is completely swamped.

Even if the transmission is error free, at a certain point the receiver will simply be unable to handle the frames as they arrive and will start to lose some.

In data link layer, flow control restricts the number of frames the sender can send before it waits for an acknowledgment from the receiver.

Approaches of Flow Control

Flow control can be broadly classified into two categories –



Feedback based Flow Control:

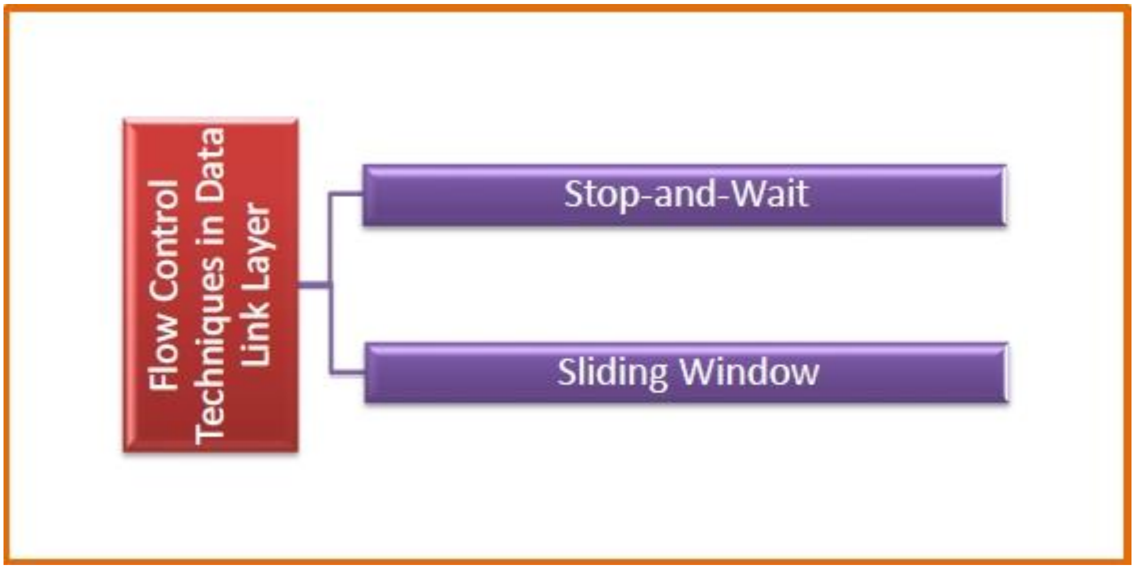
In the first one, feedback-based flow control, the receiver sends back information to the sender giving it permission to send more data or at least telling the sender how the receiver is doing.

Rate based Flow Control:

In the second one, rate-based flow control, the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver.

Flow Control Techniques in Data Link Layer:

Data link layer uses feedback based flow control mechanisms. There are two main techniques –



Stop and Wait

This protocol involves the following transitions −

- The sender sends a frame and waits for acknowledgment.
- Once the receiver receives the frame, it sends an acknowledgment frame back to the sender.

- On receiving the acknowledgment frame, the sender understands that the receiver is ready to accept the next frame. So it sender the next frame in queue.

Sliding Window

This protocol improves the efficiency of stop and waits protocol by allowing multiple frames to be transmitted before receiving an acknowledgment.

The working principle of this protocol can be described as follows −

Both the sender and the receiver have finite sized buffers called windows. The sender and the receiver agree upon the number of frames to be sent based upon the buffer size.

The sender sends multiple frames in a sequence, without waiting for acknowledgment. When its sending window is filled, it waits for acknowledgment.

On receiving acknowledgment, it advances the window and transmits the next frames, according to the number of acknowledgments received.

3.2 Error Detection and Correction:

There are many reasons such as noise, cross-talk etc., which may help data to get corrupted during transmission. The upper layers work on some generalized view of network architecture and are not aware of actual hardware data processing. Hence, the upper layers expect error-free transmission between the systems. Most of the applications would not function expectedly if they receive erroneous data. Applications such as voice and video may not be that affected and with some errors they may still function well.

Data-link layer uses error control techniques to ensure that frames, i.e. bit streams of data, are transmitted from the source to the destination with a certain extent of accuracy.

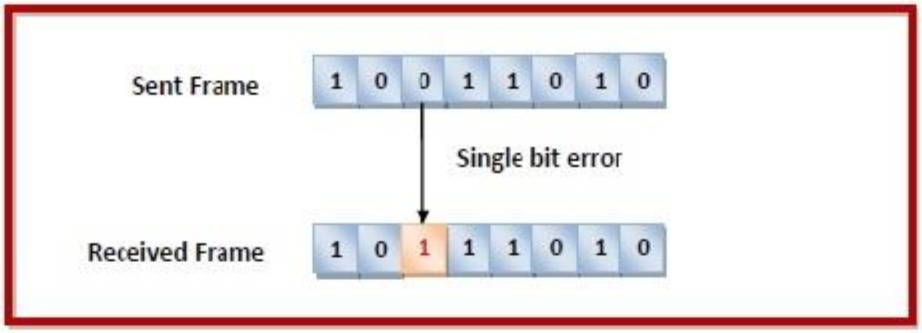
Errors

When bits are transmitted over the computer network, they are subject to get corrupted due to interference and network problems. The corrupted bits leads to spurious data being received by the destination and are called errors.

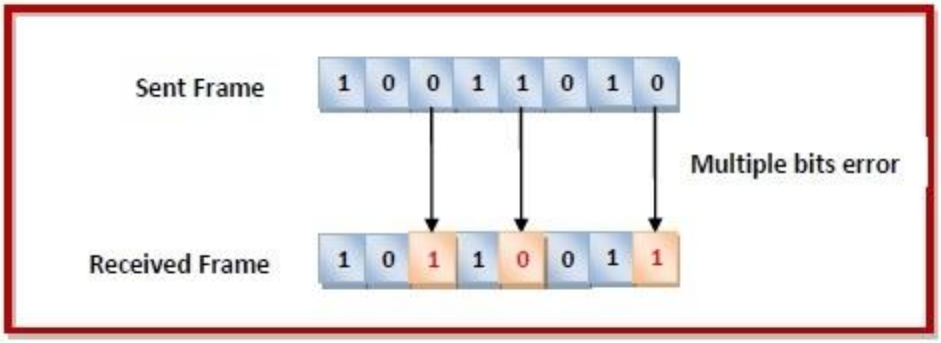
Types of Errors

Errors can be of three types, namely single bit errors, multiple bit errors, and burst errors.

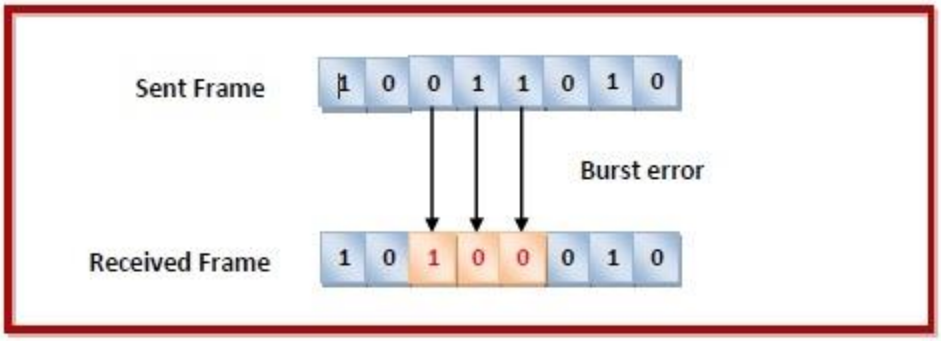
- **Single bit error** − In the received frame, only one bit has been corrupted, i.e. either changed from 0 to 1 or from 1 to 0.



- **Multiple bits error** − In the received frame, more than one bits are corrupted.



- **Burst error** − In the received frame, more than one consecutive bits are corrupted.



Error Control

Error control can be done in two ways:

- **Error detection** − Error detection involves checking whether any error has occurred or not. The number of error bits and the type of error does not matter.

- **Error correction** − Error correction involves ascertaining the exact number of bits that has been corrupted and the location of the corrupted bits.

  For both error detection and error correction, the sender needs to send some additional bits along with the data bits. The receiver performs necessary checks based upon the additional redundant bits. If it finds that the data is free from errors, it removes the redundant bits before passing the message to the upper layers.

Error Detection Techniques

There are three main techniques for detecting errors in frames:

- Parity Check
- Checksum
- Cyclic Redundancy Check (CRC).

Parity Check

The parity check is done by adding an extra bit, called parity bit to the data to make a number of 1s either even in case of even parity or odd in case of odd parity.

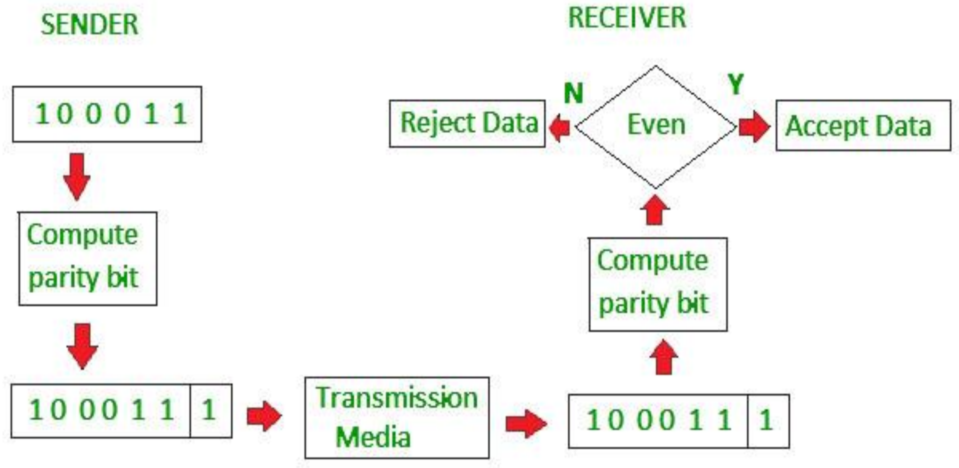
While creating a frame, the sender counts the number of 1s in it and adds the parity bit in the following way

- In case of even parity: If a number of 1s is even then parity bit value is 0. If the number of 1s is odd then parity bit value is 1.

- In case of odd parity: If a number of 1s is odd then parity bit value is 0. If a number of 1s is even then parity bit value is 1.

On receiving a frame, the receiver counts the number of 1s in it. In case of even parity check, if the count of 1s is even, the frame is accepted, otherwise, it is rejected. A similar rule is adopted for odd parity check.

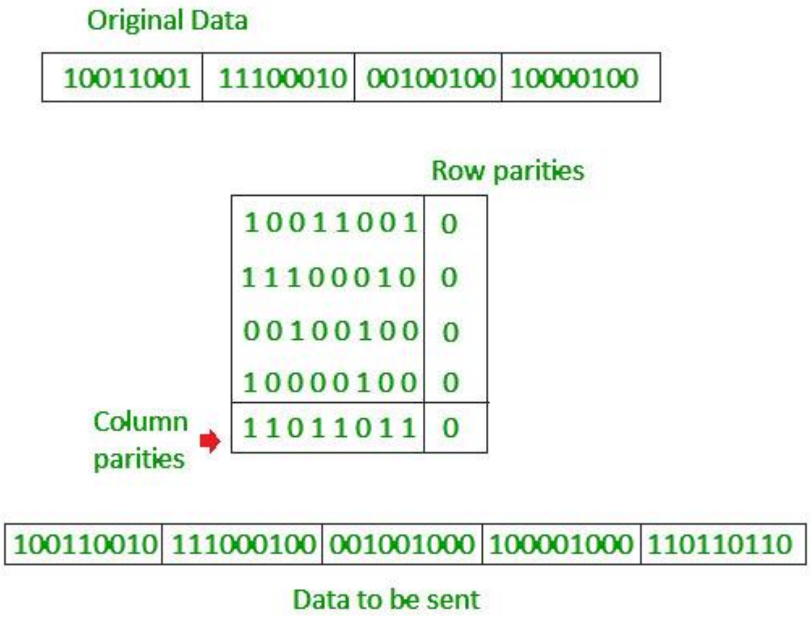
The parity check is suitable for single bit error detection only.



Two-dimensional Parity check:

Parity check bits are calculated for each row, which is equivalent to a simple parity check bit. Parity check bits are also calculated for all columns, and then both are sent along with the data.

At the receiving end these are compared with the parity bits calculated on the received data.


Original Data

10011001	11100010	00100100	10000100

Row parities

10011001	0
11100010	0
00100100	0
10000100	0
11011011	0

Column parities

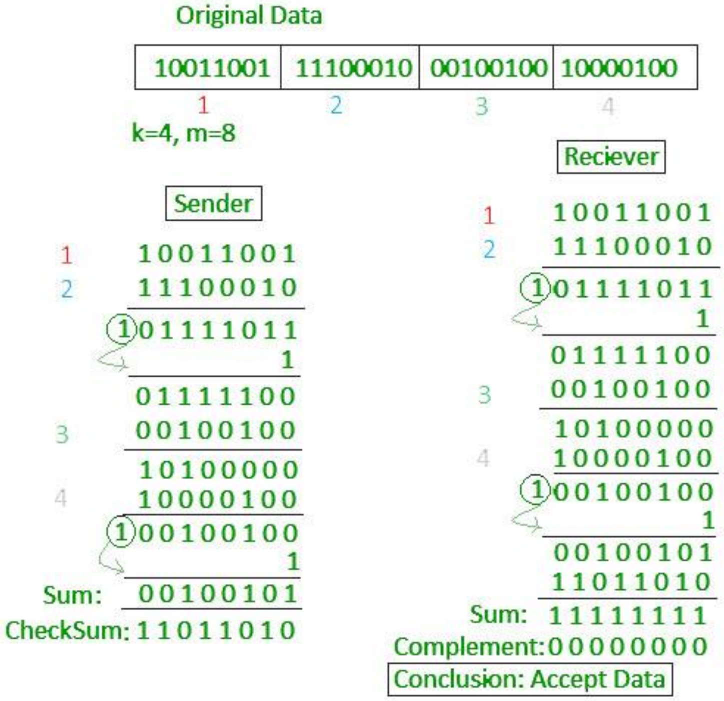
100110010	111000100	001001000	100001000	110110110

Data to be sent

Checksum

In this error detection scheme, the following procedure is applied

- Data is divided into fixed sized frames or segments.
- The sender adds the segments using 1’s complement arithmetic to get the sum. It then complements the sum to get the checksum and sends it along with the data frames.
- The receiver adds the incoming segments along with the checksum using 1’s complement arithmetic to get the sum and then complements it.
- If the result is zero, the received frames are accepted; otherwise, they are discarded.

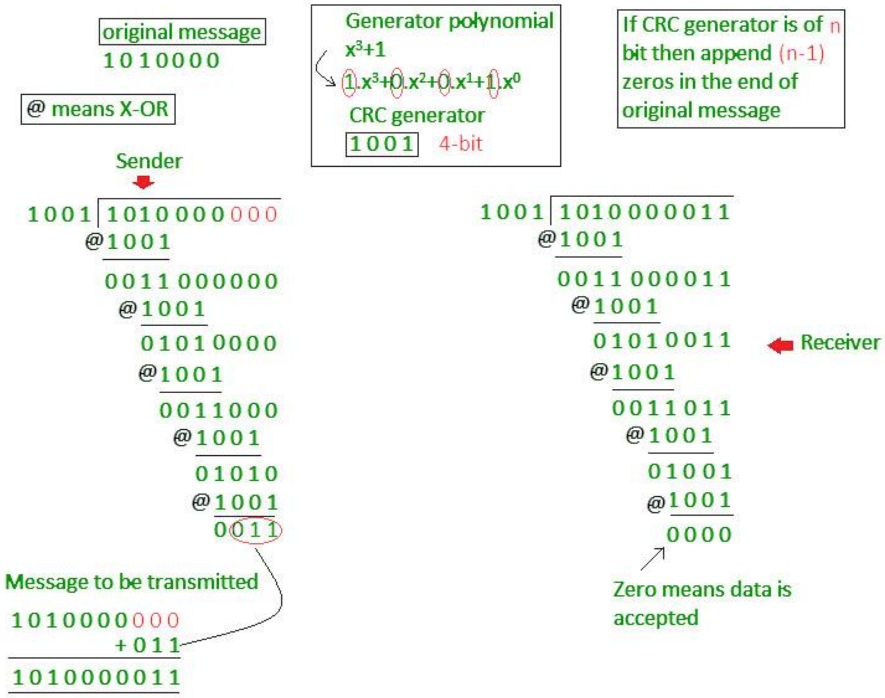


Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) involves binary division of the data bits being sent by a predetermined divisor agreed upon by the communicating system. The divisor is generated using polynomials.

- Here, the sender performs binary division of the data segment by the divisor. It then appends the remainder called CRC bits to the end of the data segment. This makes the resulting data unit exactly divisible by the divisor.

- The receiver divides the incoming data unit by the divisor. If there is no remainder, the data unit is assumed to be correct and is accepted. Otherwise, it is understood that the data is corrupted and is therefore rejected.



Error Correction Techniques

Error correction techniques find out the exact number of bits that have been corrupted and as well as their locations. There are two principle ways

- **Backward Error Correction (Retransmission)** − If the receiver detects an error in the incoming frame, it requests the sender to retransmit the frame. It is a relatively simple technique. But it can be efficiently used only where retransmitting is not expensive as in fiber optics and the time for retransmission is low relative to the requirements of the application.

- **Forward Error Correction** − If the receiver detects some error in the incoming frame, it executes error-correcting code that generates the actual frame. This saves bandwidth required for retransmission. It is inevitable in real-time systems. However, if there are too many errors, the frames need to be retransmitted.

The main error correction code is: Hamming Codes

Hamming Code:

A single additional bit can detect the error, but cannot correct it.

For correcting the errors, one has to know the exact position of the error. For example, If we want to calculate a single-bit error, the error correction code will determine which one of seven bits is in error. To achieve this, we have to add some additional redundant bits.

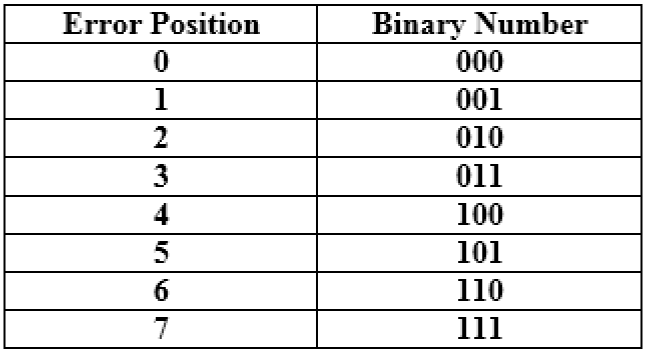
Suppose r is the number of redundant bits and d is the total number of the data bits. The number of redundant bits r can be calculated by using the formula: 2^r>=d+r+1

The value of r is calculated by using the above formula. For example, if the value of d is 4, then the possible smallest value that satisfies the above relation would be 3.

To determine the position of the bit which is in error, a technique developed by R.W Hamming is Hamming code which can be applied to any length of the data unit and uses the relationship between data units and redundant units.

Algorithm of Hamming code:

- An information of 'd' bits are added to the redundant bits 'r' to form d+r.
- The location of each of the (d+r) digits is assigned a decimal value.
- The 'r' bits are placed in the positions 1,2,.....2^{k-1}.
- At the receiving end, the parity bits are recalculated. The decimal value of the parity bits determines the position of an error.



Error Position	Binary Number
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Let's understand the concept of Hamming code through an example:

Suppose the original data is 1010 which is to be sent.

Total number of data bits’d’ = 4

Number of redundant bits r: 2^r >= d+r+1

2^r>= 4+r+1

Therefore, the value of r is 3 that satisfy the above relation.

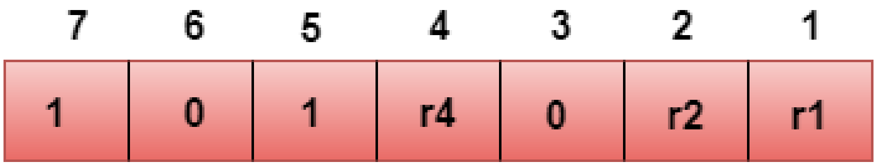
Total number of bits = d+r = 4+3 = 7;

Determining the position of the redundant bits:

The number of redundant bits is 3. The three bits are represented by r1, r2, r4. The position of the redundant bits is calculated with corresponds to the raised power of 2. Therefore, their corresponding positions are **1, 2^1, 2^2**. Therefore,

- The position of r1 = 1
- The position of r2 = 2
- The position of r4 = 4

Representation of Data on the addition of parity bits:

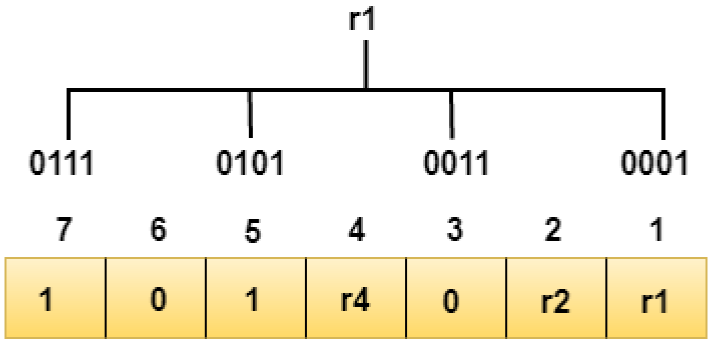


7	6	5	4	3	2	1
1	0	1	r4	0	r2	r1

Determining the Parity bits:

Determining the r1 bit

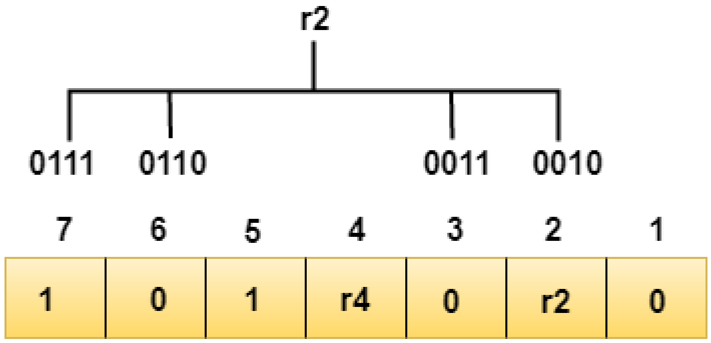
The r1 bit is calculated by performing a parity check on the bit positions whose binary representation includes 1 in the first position.



We observe from the above figure that the bit positions that include 1 in the first position are 1, 3, 5, and 7. Now, we perform the even-parity check at these bit positions. The total number of 1 at these bit positions corresponding to r1 is **even, therefore, the value of the r1 bit is 0**.

Determining r2 bit

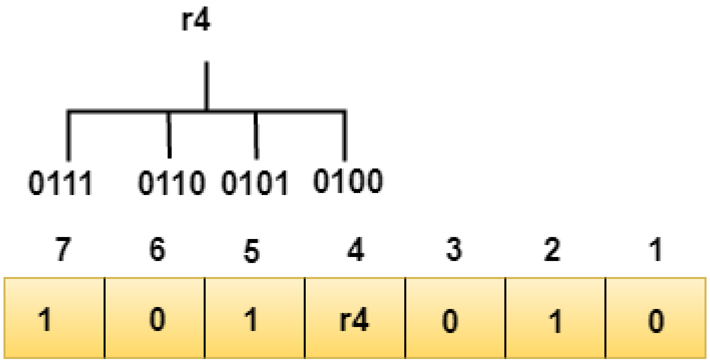
The r2 bit is calculated by performing a parity check on the bit positions whose binary representation includes 1 in the second position.



We observe from the above figure that the bit positions that include 1 in the second position are **2, 3, 6, and 7**. Now, we perform the even-parity check at these bit positions. The total number of 1 at these bit positions corresponding to r2 is **odd; therefore, the value of the r2 bit is 1**.

Determining r4 bit

The r4 bit is calculated by performing a parity check on the bit positions whose binary representation includes 1 in the third position.



We observe from the above figure that the bit positions that include 1 in the third position are **4, 5, 6, and 7**. Now, we perform the even-parity check at these bit positions. The total number of 1 at these bit positions corresponding to r4 is **even, therefore, the value of the r4 bit is 0**.

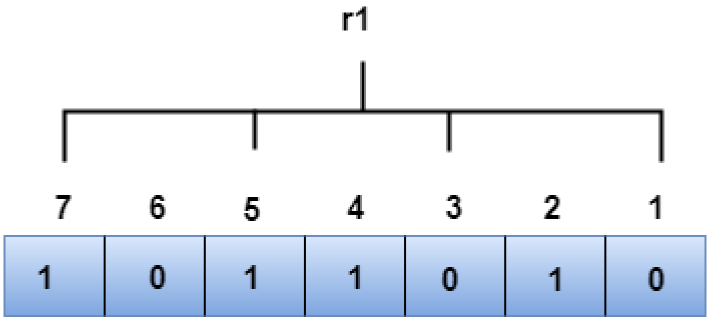
Data transferred is given below:

7	6	5	4	3	2	1
1	0	1	0	0	1	0

Suppose the 4th bit is changed from 0 to 1 at the receiving end, then parity bits are recalculated.

R1 bit

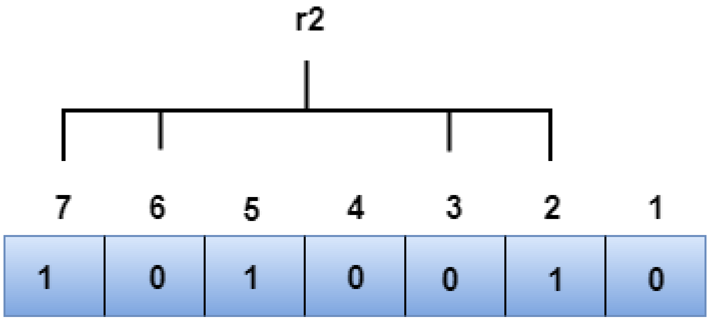
The bit positions of the r1 bit are 1,3,5,7



We observe from the above figure that the binary representation of r1 is 1100. Now, we perform the even-parity check, the total number of 1s appearing in the r1 bit is an even number. Therefore, the value of r1 is 0.

R2 bit

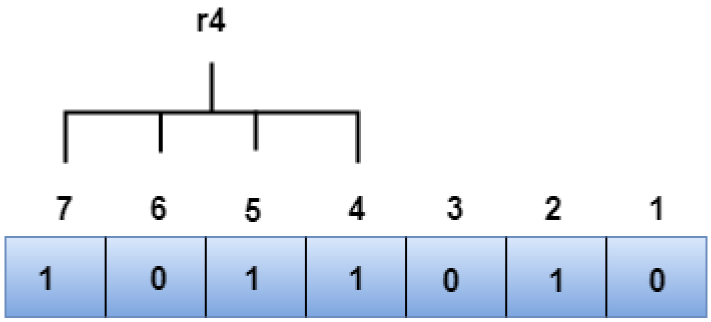
The bit positions of r2 bit are 2,3,6,7.



We observe from the above figure that the binary representation of r2 is 1001. Now, we perform the even-parity check, the total number of 1s appearing in the r2 bit is an even number. Therefore, the value of r2 is 0.

R4 bit

The bit positions of r4 bit are 4,5,6,7.



We observe from the above figure that the binary representation of r4 is 1011. Now, we perform the even-parity check, the total number of 1s appearing in the r4 bit is an odd number. Therefore, the value of r4 is 1.

- **The binary representation of redundant bits, i.e., r4r2r1 is 100, and its corresponding decimal value is 4. Therefore, the error occurs in a 4th bit position. The bit value must be changed from 1 to 0 to correct the error.**

Alpha Numeric Codes:

Alphanumeric codes are basically binary codes which are used to represent the alphanumeric data. As these codes represent data by characters, alphanumeric codes are also called “Character codes”.

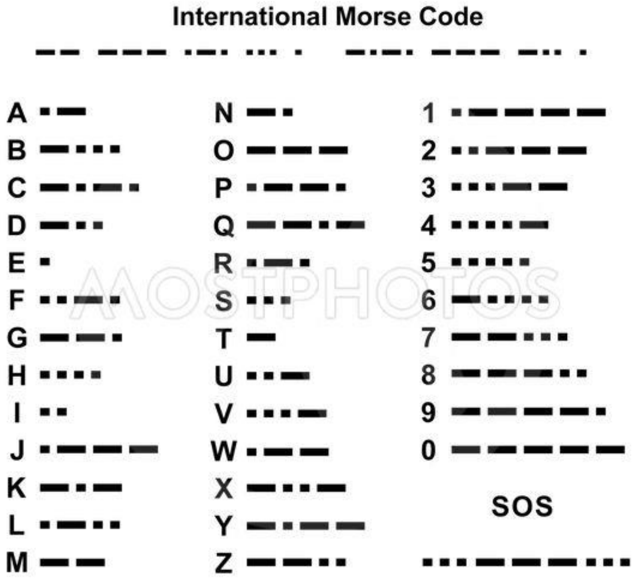
These codes can represent all types of data including alphabets, numbers, punctuation marks and mathematical symbols in the acceptable form by computers. These codes are implemented in I/O devices like key boards, monitors, printers etc.

In earlier days, punch cards are used to represent the alphanumeric codes. They are listed below as:

- MORSE code
- BAUDOT code
- HOLLERITH code
- ASCII code
- EBCDI code
- UNICODE

MORSE Code:

At the starting stage of computer and digital electronics era, Morse code is very popular and most used code. This was invented by Samuel F.B.Morse, in 1837. It was the first ever telegraphic code used in telecommunication. It is mainly used in Telegraph channels, Radio channels and in air traffic control units.



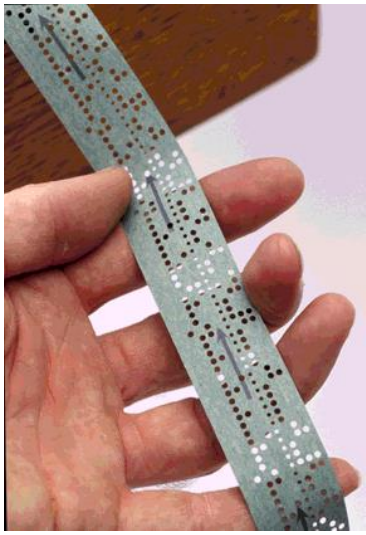
International Morse Code

−− −−− ·−· ··· ·   −·−· −−− −·· ·

A ·−	N −·	1 ·−−−−
B −···	O −−−	2 ··−−−
C −·−·	P ·−−·	3 ···−−
D −··	Q −−·−	4 ····−
E ·	R ·−·	5 ·····
F ··−·	S ···	6 −····
G −−·	T −	7 −−···
H ····	U ··−	8 −−−··
I ··	V ···−	9 −−−−·
J ·−−−	W ·−−	0 −−−−−
K −·−	X −··−	
L ·−··	Y −·−−	SOS
M −−	Z −−··	···−−−···

BOUDOT Code:

This code is invented by a French Engineer Emile Baudot, in 1870. It is a 5 unit code, means it uses 5 elements to represent an alphabet. It is also used in Telegraph networks to transfer Roman numeric.



HOLLERITH Code:

This code is developed by a company founded by Herman Hollerith in 1896. The 12 bit code used to punch cards according to the transmitting information is called “Hollerith code”.

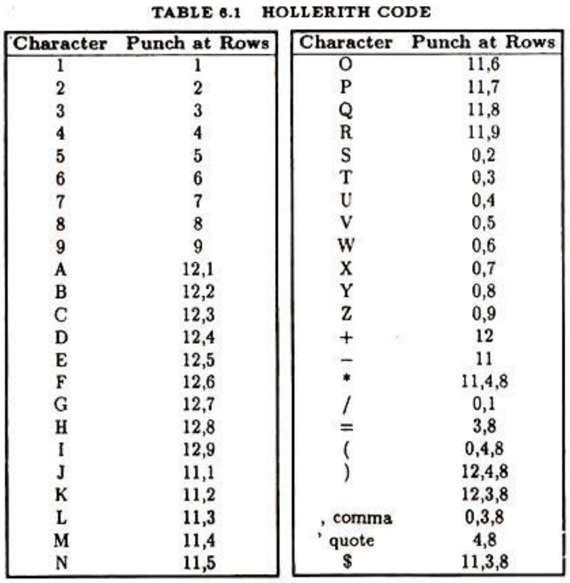


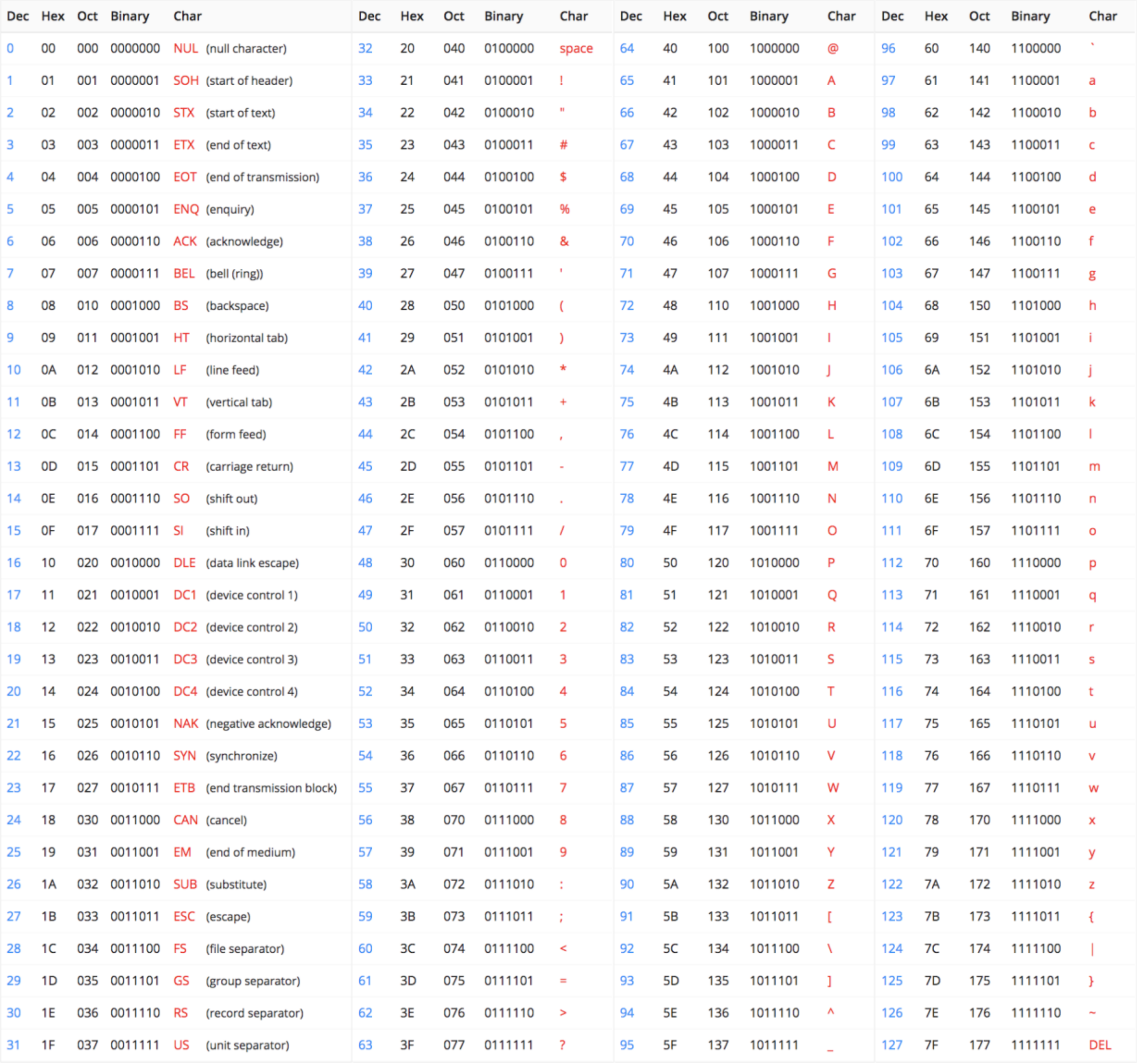
TABLE 6.1 HOLLERITH CODE

Character	Punch at Rows
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	12,1
B	12,2
C	12,3
D	12,4
E	12,5
F	12,6
G	12,7
H	12,8
I	12,9
J	11,1
K	11,2
L	11,3
M	11,4
N	11,5
O	11,6
P	11,7
Q	11,8
R	11,9
S	0,2
T	0,3
U	0,4
V	0,5
W	0,6
X	0,7
Y	0,8
Z	0,9
+	12
–	11
*	11,4,8
/	0,1
=	3,8
(	0,4,8
)	12,4,8
	12,3,8
, comma	0,3,8
’ quote	4,8
$	11,3,8

ASCII CODE:

ASCII means American Standard Code for Information Interchange. It is the world’s most popular and widely used alphanumeric code. This code was developed and first published in 1967. ASCII code is a 7 bit code that means this code uses 27 = 128 characters.

This includes 26 lower case letters (a – z), 26 upper case letters (A – Z), 33 special characters and symbols (! @ # $ etc), 33 control characters (* – + / and %) and 10 digits (0 – 9).



Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char	Dec	Hex	Oct	Binary	Char
0	00	000	0000000	NUL (null character)	32	20	040	0100000	space	64	40	100	1000000	@	96	60	140	1100000	`
1	01	001	0000001	SOH (start of header)	33	21	041	0100001	!	65	41	101	1000001	A	97	61	141	1100001	a
2	02	002	0000010	STX (start of text)	34	22	042	0100010	"	66	42	102	1000010	B	98	62	142	1100010	b
3	03	003	0000011	ETX (end of text)	35	23	043	0100011	#	67	43	103	1000011	C	99	63	143	1100011	c
4	04	004	0000100	EOT (end of transmission)	36	24	044	0100100	$	68	44	104	1000100	D	100	64	144	1100100	d
5	05	005	0000101	ENQ (enquiry)	37	25	045	0100101	%	69	45	105	1000101	E	101	65	145	1100101	e
6	06	006	0000110	ACK (acknowledge)	38	26	046	0100110	&	70	46	106	1000110	F	102	66	146	1100110	f
7	07	007	0000111	BEL (bell (ring))	39	27	047	0100111	'	71	47	107	1000111	G	103	67	147	1100111	g
8	08	010	0001000	BS (backspace)	40	28	050	0101000	(	72	48	110	1001000	H	104	68	150	1101000	h
9	09	011	0001001	HT (horizontal tab)	41	29	051	0101001	)	73	49	111	1001001	I	105	69	151	1101001	i
10	0A	012	0001010	LF (line feed)	42	2A	052	0101010	*	74	4A	112	1001010	J	106	6A	152	1101010	j
11	0B	013	0001011	VT (vertical tab)	43	2B	053	0101011	+	75	4B	113	1001011	K	107	6B	153	1101011	k
12	0C	014	0001100	FF (form feed)	44	2C	054	0101100	,	76	4C	114	1001100	L	108	6C	154	1101100	l
13	0D	015	0001101	CR (carriage return)	45	2D	055	0101101	-	77	4D	115	1001101	M	109	6D	155	1101101	m
14	0E	016	0001110	SO (shift out)	46	2E	056	0101110	.	78	4E	116	1001110	N	110	6E	156	1101110	n
15	0F	017	0001111	SI (shift in)	47	2F	057	0101111	/	79	4F	117	1001111	O	111	6F	157	1101111	o
16	10	020	0010000	DLE (data link escape)	48	30	060	0110000	0	80	50	120	1010000	P	112	70	160	1110000	p
17	11	021	0010001	DC1 (device control 1)	49	31	061	0110001	1	81	51	121	1010001	Q	113	71	161	1110001	q
18	12	022	0010010	DC2 (device control 2)	50	32	062	0110010	2	82	52	122	1010010	R	114	72	162	1110010	r
19	13	023	0010011	DC3 (device control 3)	51	33	063	0110011	3	83	53	123	1010011	S	115	73	163	1110011	s
20	14	024	0010100	DC4 (device control 4)	52	34	064	0110100	4	84	54	124	1010100	T	116	74	164	1110100	t
21	15	025	0010101	NAK (negative acknowledge)	53	35	065	0110101	5	85	55	125	1010101	U	117	75	165	1110101	u
22	16	026	0010110	SYN (synchronize)	54	36	066	0110110	6	86	56	126	1010110	V	118	76	166	1110110	v
23	17	027	0010111	ETB (end transmission block)	55	37	067	0110111	7	87	57	127	1010111	W	119	77	167	1110111	w
24	18	030	0011000	CAN (cancel)	56	38	070	0111000	8	88	58	130	1011000	X	120	78	170	1111000	x
25	19	031	0011001	EM (end of medium)	57	39	071	0111001	9	89	59	131	1011001	Y	121	79	171	1111001	y
26	1A	032	0011010	SUB (substitute)	58	3A	072	0111010	:	90	5A	132	1011010	Z	122	7A	172	1111010	z
27	1B	033	0011011	ESC (escape)	59	3B	073	0111011	;	91	5B	133	1011011	[	123	7B	173	1111011	{
28	1C	034	0011100	FS (file separator)	60	3C	074	0111100	<	92	5C	134	1011100	\	124	7C	174	1111100	|
29	1D	035	0011101	GS (group separator)	61	3D	075	0111101	=	93	5D	135	1011101	]	125	7D	175	1111101	}
30	1E	036	0011110	RS (record separator)	62	3E	076	0111110	>	94	5E	136	1011110	^	126	7E	176	1111110	~
31	1F	037	0011111	US (unit separator)	63	3F	077	0111111	?	95	5F	137	1011111	_	127	7F	177	1111111	DEL

Example:

If we want to print the name LONDAN, the ASCII code is?

The ASCII-7 equivalent of L = 100 1100

The ASCII-7 equivalent of O = 100 1111

The ASCII-7 equivalent of N = 100 1110

The ASCII-7 equivalent of D = 100 0100

The ASCII-7 equivalent of A = 100 0001

The ASCII-7 equivalent of N = 100 1110

The output of LONDAN in ASCII code is 1 0 0 1 1 0 0 1 0 0 1 1 1 1 1 0 0 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 1 1 0.

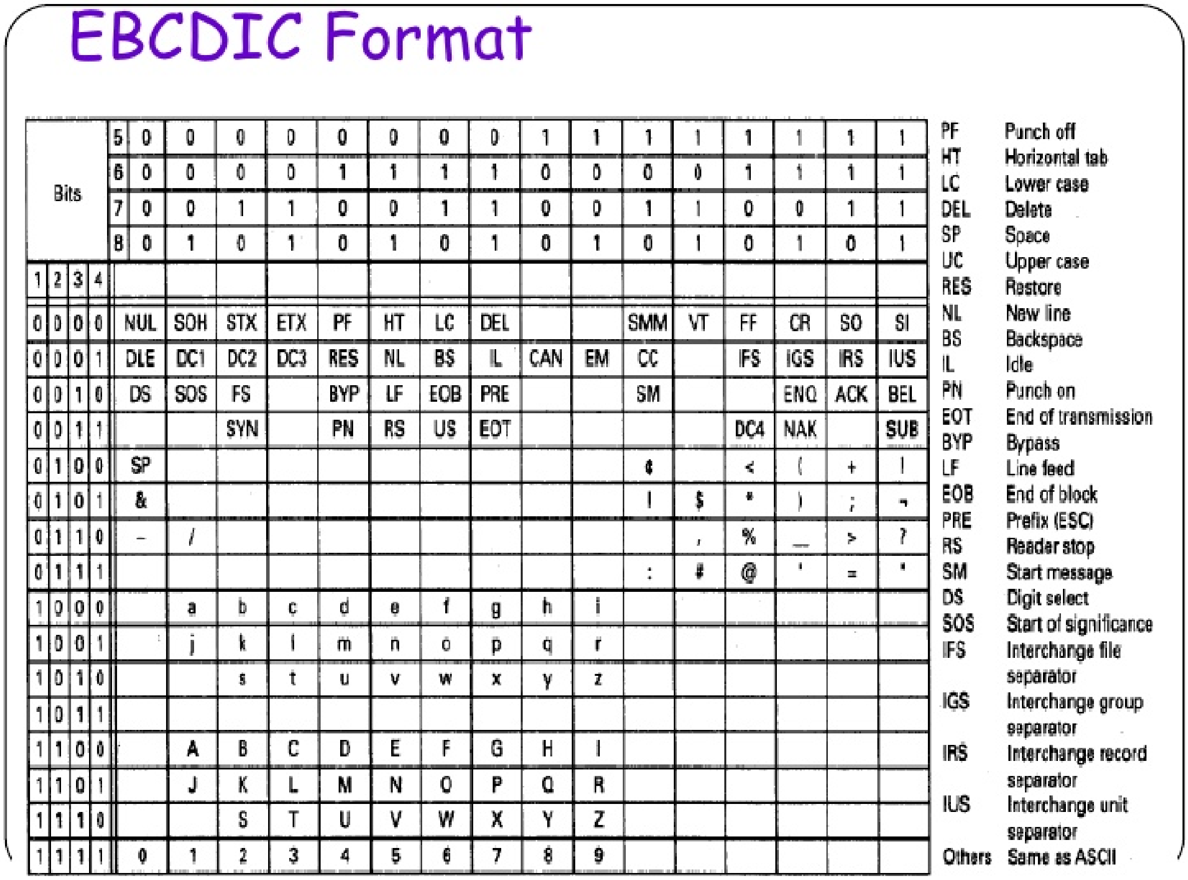
EBCDI CODE:

EBCDI stands for Extended Binary Coded Decimal Interchange code. This code is developed by IBM Inc Company.

It is an 8 bit code, so we can represent 28 = 256 characters by using EBCDI code.

This include all the letters and symbols like 26 lower case letters (a – z), 26 upper case letters (A – Z), 33 special characters and symbols (! @ # $ etc), 33 control characters (* – + / and % etc) and 10 digits (0 – 9).

In the EBCDI code, the 8 bit code the numbers are represented by 8421 BCD code preceded by 1111.



EBCDIC Format

Bits				5	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				6	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				7	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				8	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	2	3	4																	
0	0	0	0		NUL	SOH	STX	ETX	PF	HT	LC	DEL			SMM	VT	FF	CR	SO	SI
0	0	0	1		DLE	DC1	DC2	DC3	RES	NL	BS	IL	CAN	EM	CC		IFS	IGS	IRS	IUS
0	0	1	0		DS	SOS	FS		BYP	LF	EOB	PRE			SM			ENQ	ACK	BEL
0	0	1	1				SYN		PN	RS	US	EOT					DC4	NAK		SUB
0	1	0	0		SP										¢		<	(	+	|
0	1	0	1		&										!	$	*	)	;	¬
0	1	1	0		–	/										,	%	_	>	?
0	1	1	1												:	#	@	'	=	"
1	0	0	0			a	b	c	d	e	f	g	h	i						
1	0	0	1			j	k	l	m	n	o	p	q	r						
1	0	1	0				s	t	u	v	w	x	y	z						
1	0	1	1																	
1	1	0	0			A	B	C	D	E	F	G	H	I						
1	1	0	1			J	K	L	M	N	O	P	Q	R						
1	1	1	0				S	T	U	V	W	X	Y	Z						
1	1	1	1		0	1	2	3	4	5	6	7	8	9						

PF	Punch off
HT	Horizontal tab
LC	Lower case
DEL	Delete
SP	Space
UC	Upper case
RES	Restore
NL	New line
BS	Backspace
IL	Idle
PN	Punch on
EOT	End of transmission
BYP	Bypass
LF	Line feed
EOB	End of block
PRE	Prefix (ESC)
RS	Reader stop
SM	Start message
DS	Digit select
SOS	Start of significance
IFS	Interchange file separator
IGS	Interchange group separator
IRS	Interchange record separator
IUS	Interchange unit separator
Others	Same as ASCII

UNICODE:

The draw backs in ASCII code and EBCDI code are that they are not compatible to all languages and they do not have sufficient set of characters to represent all types of data.

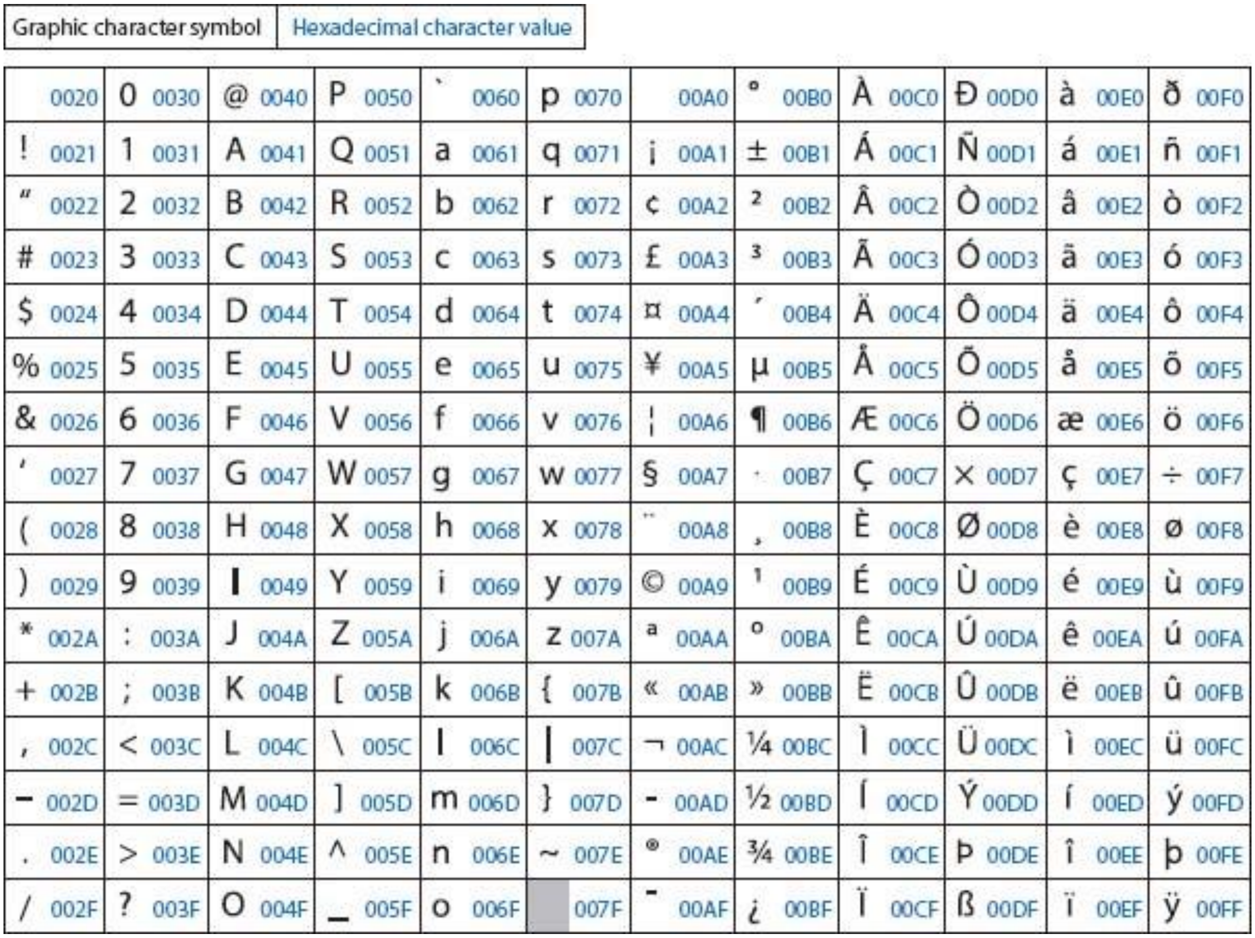
To overcome these drawback this UNICODE is developed.

UNICODE is the new concept of all digital coding techniques. In this we have a different character to represent every number.

It is the most advanced and sophisticated language with the ability to represent any type of data.

SO this is known as “Universal code”. It is a 16 bit code, with which we can represent 216 = 65536 different characters.

UNICODE is developed by the combined effort of UNICODE consortium and ISO (International organization for Standardization).



Graphic character symbol	Hexadecimal character value

0020	0 0030	@ 0040	P 0050	` 0060	p 0070	00A0	° 00B0	À 00C0	Đ 00D0	à 00E0	ð 00F0
! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
" 0022	2 0032	B 0042	R 0052	b 0062	r 0072	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
' 0027	7 0037	G 0047	W 0057	g 0067	w 0077	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
(0028	8 0038	H 0048	X 0058	h 0068	x 0078	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
, 002C	< 003C	L 004C	\ 005C	l 006C	| 007C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
- 002D	= 003D	M 004D	] 005D	m 006D	} 007D	- 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
/ 002F	? 003F	O 004F	_ 005F	o 006F	007F	¯ 00AF	¿ 00BF	Ï 00CF	ß 00DF	ï 00EF	ÿ 00FF

Elementary Data Link Protocols:

When the data link layer accepts a packet, it encapsulates the packet in a frame by adding a data link header and trailer to it. Thus, a frame consists of an **embedded packet**, **control information** (in the header), and a **checksum** (in the trailer).

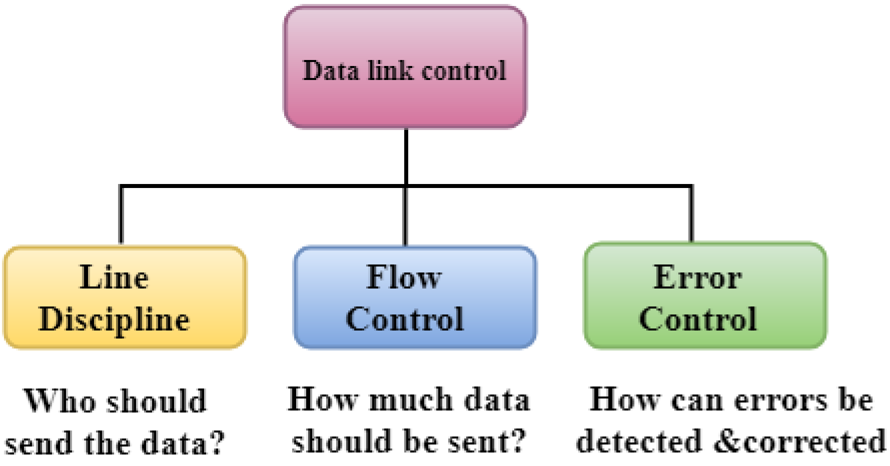
The frame is then transmitted to the data link layer on the other machine. We will assume that there exist suitable library procedures to physical layer to send and receive a frame.

Data Link Controls

Data Link Control is the service provided by the Data Link Layer to provide reliable data transfer over the physical medium. For example, In the half-duplex transmission mode, one device can only transmit the data at a time. If both the devices at the end of the links transmit the data simultaneously, they will collide and leads to the loss of the information. The Data link layer provides the coordination among the devices so that no collision occurs.

The Data link layer provides three functions:

- Line discipline
- Flow Control
- Error Control


Data link control
Line Discipline
Flow Control
Error Control
Who should send the data?
How much data should be sent?
How can errors be detected &corrected

Line Discipline

- Line Discipline is a functionality of the Data link layer that provides the coordination among the link systems. It determines which device can send, and when it can send the data.

Line Discipline can be achieved in two ways:

- ENQ/ACK
- Poll/select

END/ACK

END/ACK stands for Enquiry/Acknowledgement is used when there is no wrong receiver available on the link and having a dedicated path between the two devices so that the device capable of receiving the transmission is the intended one.

END/ACK coordinates which device will start the transmission and whether the recipient is ready or not.

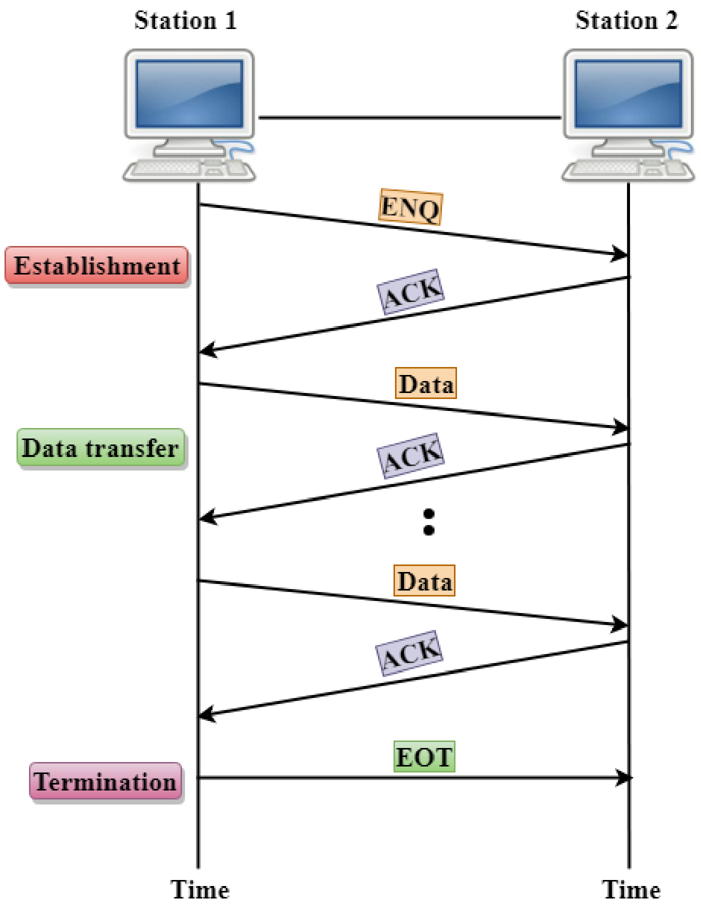
Working of END/ACK

The transmitter transmits the frame called an Enquiry (ENQ) asking whether the receiver is available to receive the data or not.

The receiver responses either with the positive acknowledgement(ACK) or with the negative acknowledgement(NACK) where positive acknowledgement means that the receiver is ready to receive the transmission and negative acknowledgement means that the receiver is unable to accept the transmission.

Following are the responses of the receiver:

- If the response to the ENQ is positive, the sender will transmit its data, and once all of its data has been transmitted, the device finishes its transmission with an EOT (END-of-Transmission) frame.
- If the response to the ENQ is negative, then the sender disconnects and restarts the transmission at another time.
- If the response is neither negative nor positive, the sender assumes that the ENQ frame was lost during the transmission and makes three attempts to establish a link before giving up.



Poll/Select

The Poll/Select method of line discipline works with those topologies where one device is designated as a primary station, and other devices are secondary stations.

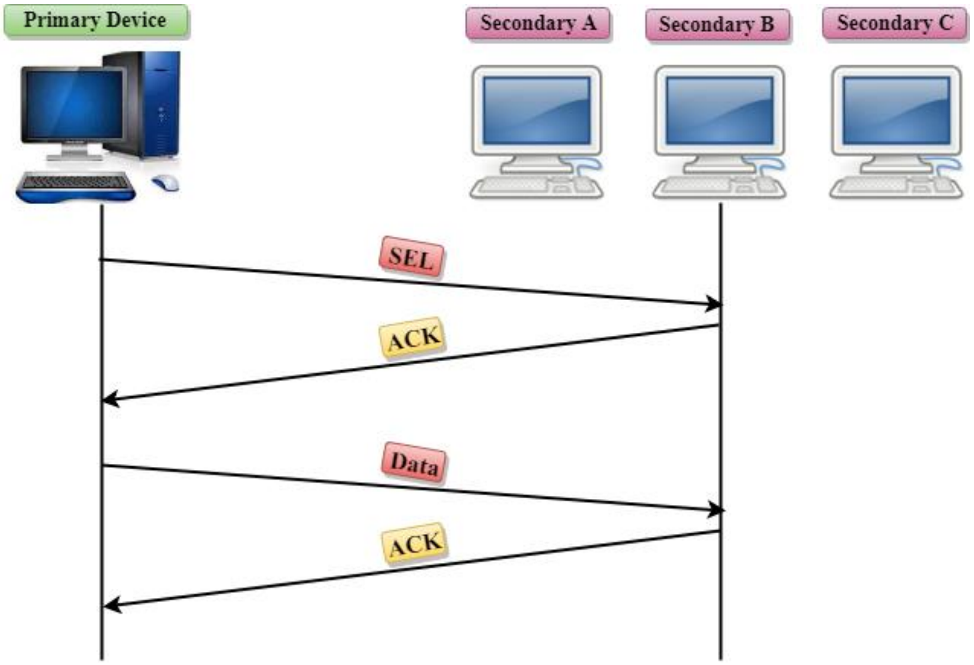
Working of Poll/Select

- In this, the primary device and multiple secondary devices consist of a single transmission line, and all the exchanges are made through the primary device even though the destination is a secondary device.
- The primary device has control over the communication link, and the secondary device follows the instructions of the primary device.

- The primary device determines which device is allowed to use the communication channel. Therefore, we can say that it is an initiator of the session.
- If the primary device wants to receive the data from the secondary device, it asks the secondary device that they anything to send, this process is known as polling.
- If the primary device wants to send some data to the secondary device, then it tells the target secondary to get ready to receive the data, this process is known as selecting.

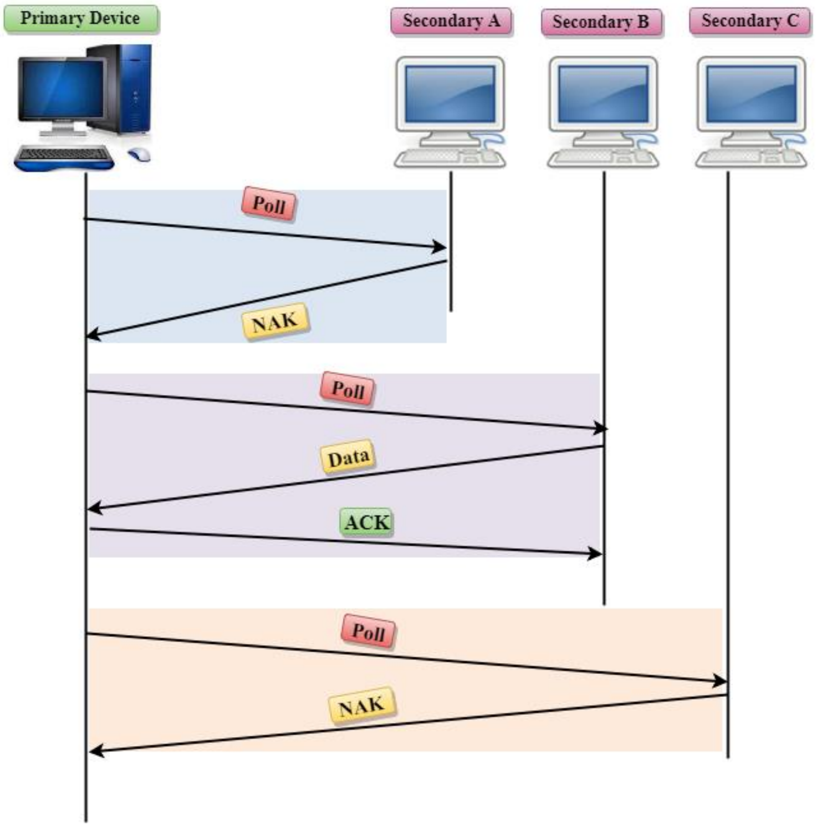
Select

- The select mode is used when the primary device has something to send.
- When the primary device wants to send some data, then it alerts the secondary device for the upcoming transmission by transmitting a Select (SEL) frame, one field of the frame includes the address of the intended secondary device.
- When the secondary device receives the SEL frame, it sends an acknowledgement that indicates the secondary ready status.
- If the secondary device is ready to accept the data, then the primary device sends two or more data frames to the intended secondary device. Once the data has been transmitted, the secondary sends an acknowledgement specifies that the data has been received.



Poll

- The Poll mode is used when the primary device wants to receive some data from the secondary device.
- When a primary device wants to receive the data, then it asks each device whether it has anything to send.
- Firstly, the primary asks (poll) the first secondary device, if it responds with the NACK (Negative Acknowledgement) means that it has nothing to send. Now, it approaches the second secondary device, it responds with the ACK means that it has the data to send. The secondary device can send more than one frame one after another or sometimes it may be required to send ACK before sending each one, depending on the type of the protocol being used.



Elementary Data Link Protocols: When a frame arrives at the receiver, the hardware computes the checksum. If the checksum is incorrect, the data link layer is so informed (event = cksum err).

If the inbound frame arrived undamaged, thedata link layer is also informed (event = frame arrival) so that it can acquire the frame for inspection using from physical layer.

As soon as the receiving data link layer has acquired an undamaged frame, it checks the control information in the header, and if everything is all right, passes the packet portion to the network layer. Under no circumstances is a frame header ever given to a network layer.

There is a good reason why the network layer must never be given any part of the frame header: to keep the network and data link protocols completely separate.

Five data structures are defined there:

1. Boolean
2. seq_nr
3. packet
4. frame_kind
5. frame

A **Boolean** is an enumerated type and can take on the values true and false.

A **seq_nr** is a small integer used to number the frames. These sequence numbers run from 0 up to and including MAX_SEQ, which is defined in each protocol needing it.

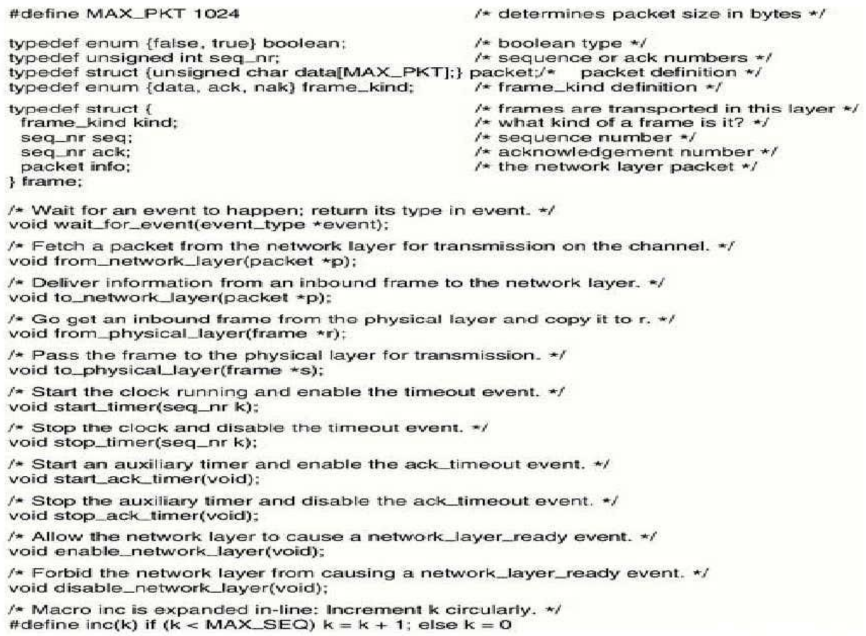
A **packet** is the unit of information exchanged between the network layer and the data link layer on the same machine, or between network layer peers.

In our model it always contains **MAX_PKT** bytes, but more realistically it would be of variable length.

A frame is composed of four fields: **kind, seq, ack,** and **info**, the first three of which contain control information and the last of which may contain actual data to be transferred.

These control fields are collectively called the **frame header**.

The **kind** field tells whether there are any data in the frame. The procedures **to network layer** and **from network layer** are used by the data link layer to pass packets to the network layer and accept packets from the network layer respectively.



```
#define MAX_PKT 1024                                  /* determines packet size in bytes */

typedef enum {false, true} boolean;                   /* boolean type */
typedef unsigned int seq_nr;                          /* sequence or ack numbers */
typedef struct {unsigned char data[MAX_PKT];} packet;/*    packet definition */
typedef enum {data, ack, nak} frame_kind;             /* frame_kind definition */

typedef struct {                                      /* frames are transported in this layer */
  frame_kind kind;                                    /* what kind of a frame is it? */
  seq_nr seq;                                         /* sequence number */
  seq_nr ack;                                         /* acknowledgement number */
  packet info;                                        /* the network layer packet */
} frame;

/* Wait for an event to happen; return its type in event. */
void wait_for_event(event_type *event);

/* Fetch a packet from the network layer for transmission on the channel. */
void from_network_layer(packet *p);

/* Deliver information from an inbound frame to the network layer. */
void to_network_layer(packet *p);

/* Go get an inbound frame from the physical layer and copy it to r. */
void from_physical_layer(frame *r);

/* Pass the frame to the physical layer for transmission. */
void to_physical_layer(frame *s);

/* Start the clock running and enable the timeout event. */
void start_timer(seq_nr k);

/* Stop the clock and disable the timeout event. */
void stop_timer(seq_nr k);

/* Start an auxiliary timer and enable the ack_timeout event. */
void start_ack_timer(void);

/* Stop the auxiliary timer and disable the ack_timeout event. */
void stop_ack_timer(void);

/* Allow the network layer to cause a network_layer_ready event. */
void enable_network_layer(void);

/* Forbid the network layer from causing a network_layer_ready event. */
void disable_network_layer(void);

/* Macro inc is expanded in-line: Increment k circularly. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

In most of the protocols, we assume that the channel is unreliable and loses entire frames upon occasion. To be able to recover from such calamities, the sending data link layer must start an internal timer or clock whenever it sends a frame.

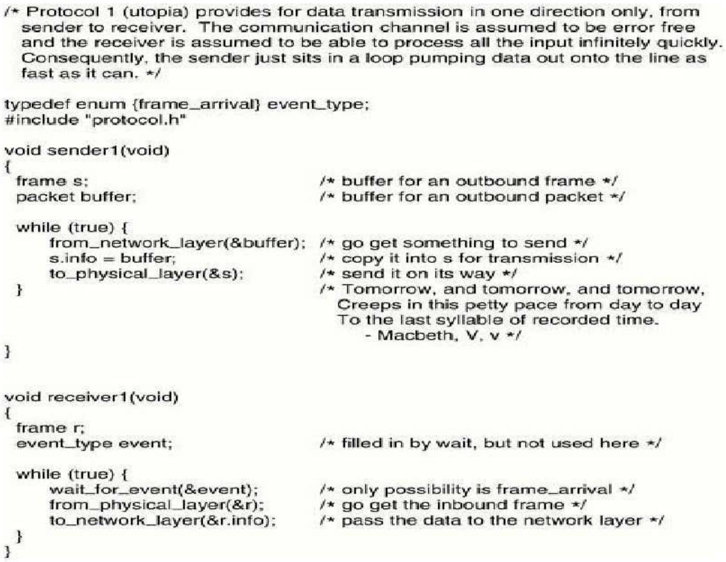
If no reply has been received within a certain predetermined time interval, the clock timeout and the data link layer receive an interrupt signal.

Three different protocols have been proposed to demonstrate the same process on different channels.

1. An Unrestricted Simplex Protocol.
2. A Simplex Stop-and-Wait Protocol.
3. A Simplex Protocol for a Noisy Channel.

An Unrestricted Simplex Protocol

As an initial example we will consider a protocol that is as simple as it can be. Data are transmitted in one direction only. Both the transmitting and receiving network layers are always ready. Processing time can be ignored. Infinite buffer space is available. And best of all, the communication channel between the data link layers never damages or loses frames.



```
/* Protocol 1 (utopia) provides for data transmission in one direction only, from
   sender to receiver.  The communication channel is assumed to be error free
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                              /* buffer for an outbound frame */
  packet buffer;                        /* buffer for an outbound packet */

  while (true) {
       from_network_layer(&buffer);     /* go get something to send */
       s.info = buffer;                 /* copy it into s for transmission */
       to_physical_layer(&s);           /* send it on its way */
  }                                     /* Tomorrow, and tomorrow, and tomorrow,
                                           Creeps in this petty pace from day to day
                                           To the last syllable of recorded time.
                                               - Macbeth, V, v */
}


void receiver1(void)
{
  frame r;
  event_type event;                     /* filled in by wait, but not used here */

  while (true) {
       wait_for_event(&event);          /* only possibility is frame_arrival */
       from_physical_layer(&r);         /* go get the inbound frame */
       to_network_layer(&r.info);       /* pass the data to the network layer */
  }
}
```

The protocol consists of two distinct procedures, a sender and a receiver. The sender runs in the data link layer of the source machine, and the receiver runs in the data link layer of the destination machine. No sequence numbers or acknowledgements are used here, so MAX_SEQ is not needed. The only event type possible is frame_arrival (i.e., the arrival of an undamaged frame).

The sender is in an infinite while loop just pumping data out onto the line as fast as it can.

The body of the loop consists of three actions:

1. Go fetch a packet from the (always obliging) network layer
2. Construct an outbound frame using the variable s
3. Send the frame on its way.

Only the info field of the frame is used by this protocol, because the other fields have to do with error and flow control and there are no errors or flow control restrictions here.

The receiver is equally simple. Initially, it waits for something to happen, the only possibility being the arrival of an undamaged frame.

Eventually, the frame arrives and the procedure wait_for_event returns, with event set to frame_arrival (which is ignored anyway).

The call to from_physical_layer removes the newly arrived frame from the hardware buffer and puts it in the variable r, where the receiver code can get at it. Finally, the data portion is passed on to the network layer, and the data link layer settles back to wait for the next frame, effectively suspending itself until the frame arrives.

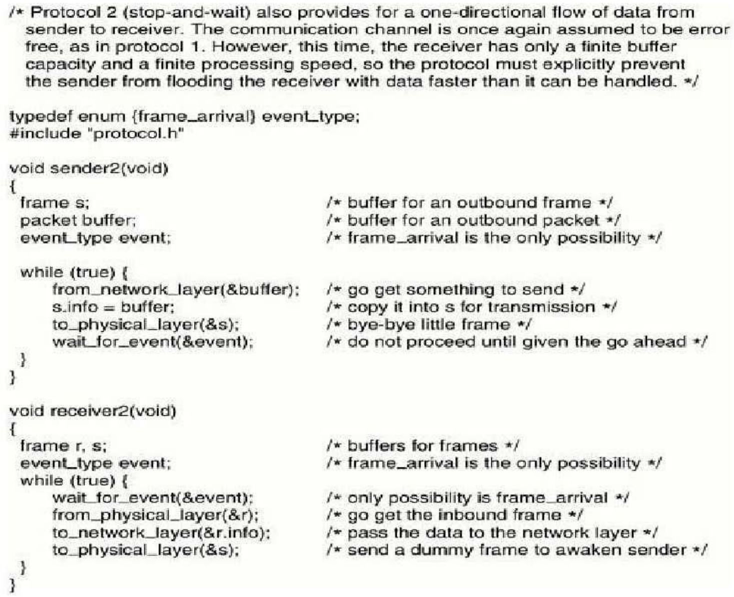
A Simplex Stop-and-Wait Protocol

Now we will drop the most unrealistic restriction used in protocol 1: the ability of the receiving network layer to process incoming data infinitely quickly (or equivalently, the presence in the receiving data link layer of an infinite amount of buffer space in which to store all incoming frames while they are waiting their respective turns). The communication channel is still assumed to be error free however, and the data traffic is still simplex.

The main problem we have to deal with here is how to prevent the sender from flooding the receiver with data faster than the latter is able to process them. In essence, if the receiver requires a time (t) to execute from physical layer plus to network layer, the sender must transmit at an average rate less than one frame per time (t).

A more general solution to this dilemma is to have the receiver provide feedback to the sender. After having passed a packet to its network layer, the receiver sends a little dummy frame back to the sender which, in effect, gives the sender permission to transmit the next frame. After having sent a frame, the sender is required by the protocol to bide its time until the little dummy (i.e., acknowledgement) frame arrives. Using feedback from the receiver to let the sender know when it may send more data is an example of the flow control mentioned earlier.

Protocols in which the sender sends one frame and then waits for an acknowledgement before proceeding are called stop-and-wait.



```
/* Protocol 2 (stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time, the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                              /* buffer for an outbound frame */
  packet buffer;                        /* buffer for an outbound packet */
  event_type event;                     /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);      /* go get something to send */
      s.info = buffer;                  /* copy it into s for transmission */
      to_physical_layer(&s);            /* bye-bye little frame */
      wait_for_event(&event);           /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                           /* buffers for frames */
  event_type event;                     /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event);           /* only possibility is frame_arrival */
      from_physical_layer(&r);          /* go get the inbound frame */
      to_network_layer(&r.info);        /* pass the data to the network layer */
      to_physical_layer(&s);            /* send a dummy frame to awaken sender */
  }
}
```

unlike in protocol 1, the sender must wait until an acknowledgement frame arrives before looping back and fetching the next packet from the network layer. The sending data link layer need not even inspect the incoming frame: there is only one possibility. The incoming frame is always an acknowledgement.

The only difference between receiver1 and receiver2 is that after delivering a packet to the network layer, receiver2 sends an acknowledgement frame back to the sender before entering the wait loop again. Because only the arrival of the frame back at the sender is important, not its contents, the receiver need not put any particular information in it.

A Simplex Protocol for a Noisy Channel

Now let us consider the normal situation of a communication channel that makes errors. Frames may be either damaged or lost completely. However, we assume that if a frame is damaged in transit, the receiver hardware will detect this when it computes the checksum.

If the frame is damaged in such a way that the checksum is nevertheless correct, an unlikely occurrence, this protocol (and all other protocols) can fail (i.e., deliver an incorrect packet to the network layer).

At first glance it might seem that a variation of protocol 2 would work: adding a timer. The sender could send a frame, but the receiver would only send an acknowledgement frame if the data were correctly received. If a damaged frame arrived at the receiver, it would be discarded.

After a while the sender would time out and send the frame again. This process would be repeated until the frame finally arrived intact.

The above scheme has a fatal flaw in it. Think about the problem and try to discover what might go wrong before reading further.

To see what might go wrong, remember that it is the task of the data link layer processes to provide error-free, transparent communication between network layer processes.

The network layer on machine A gives a series of packets to its data link layer, which must ensure that an identical series of packets are delivered to the network layer on machine B by its data link layer.

In particular, the network layer on B has no way of knowing that a packet has been lost or duplicated, so the data link layer must guarantee that no combination of transmission errors, however unlikely, can cause a duplicate packet to be delivered to a network layer.

Consider the following scenario:

1. The network layer on A gives packet 1 to its data link layer. The packet is correctly received at B and passed to the network layer on B. B sends an acknowledgement frame back to A.

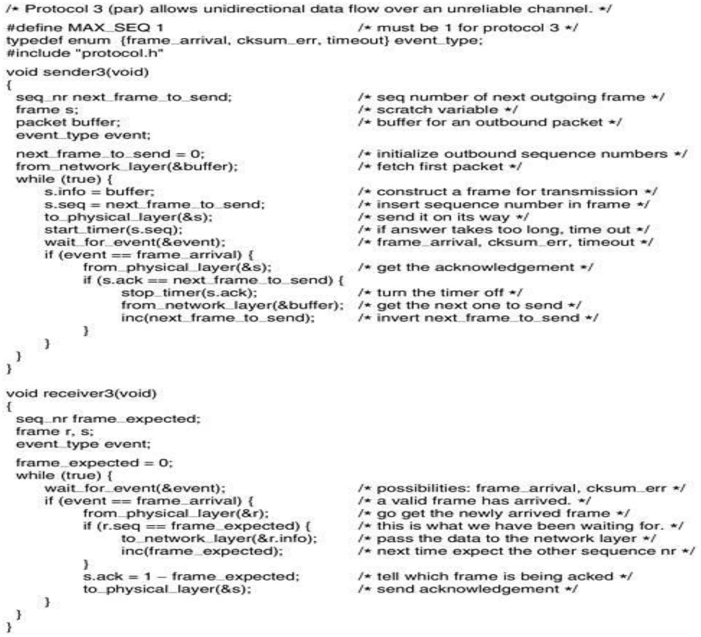
2. The acknowledgement frame gets lost completely. It just never arrives at all. Life would be a great deal simpler if the channel mangled and lost only data frames and not control frames, but sad to say, the channel is not very discriminating.

3. The data link layer on A eventually times out. Not having received an acknowledgement, it (incorrectly) assumes that its data frame was lost or damaged and sends the frame containing packet 1 again.

4. The duplicate frame also arrives at the data link layer on B perfectly and is unwittingly passed to the network layer there. If A is sending a file to B, part of the file will be duplicated (i.e., the copy of the file made by B will be incorrect and the error will not have been detected). In other words, the protocol will fail.

Clearly, what is needed is some way for the receiver to be able to distinguish a frame that it is seeing for the first time from a retransmission. The obvious way to achieve this is to have the sender put a sequence number in the header of each frame it sends. Then the receiver can check the sequence number of each arriving frame to see if it is a new frame or a duplicate to be discarded.

Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called PAR (Positive Acknowledgement with Retransmission) or ARQ (Automatic Repeat Request). Like protocol 2, this one also transmits data only in one direction.



```
/* Protocol 3 (par) allows unidirectional data flow over an unreliable channel. */
#define MAX_SEQ 1                           /* must be 1 for protocol 3 */
typedef enum  {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
 seq_nr next_frame_to_send;                 /* seq number of next outgoing frame */
 frame s;                                   /* scratch variable */
 packet buffer;                             /* buffer for an outbound packet */
 event_type event;

 next_frame_to_send = 0;                    /* initialize outbound sequence numbers */
 from_network_layer(&buffer);               /* fetch first packet */
 while (true) {
     s.info = buffer;                       /* construct a frame for transmission */
     s.seq = next_frame_to_send;            /* insert sequence number in frame */
     to_physical_layer(&s);                 /* send it on its way */
     start_timer(s.seq);                    /* if answer takes too long, time out */
     wait_for_event(&event);                /* frame_arrival, cksum_err, timeout */
     if (event == frame_arrival) {
          from_physical_layer(&s);          /* get the acknowledgement */
          if (s.ack == next_frame_to_send) {
               stop_timer(s.ack);           /* turn the timer off */
               from_network_layer(&buffer); /* get the next one to send */
               inc(next_frame_to_send);     /* invert next_frame_to_send */
          }
     }
 }
}

void receiver3(void)
{
 seq_nr frame_expected;
 frame r, s;
 event_type event;

 frame_expected = 0;
 while (true) {
     wait_for_event(&event);                /* possibilities: frame_arrival, cksum_err */
     if (event == frame_arrival) {          /* a valid frame has arrived. */
          from_physical_layer(&r);          /* go get the newly arrived frame */
          if (r.seq == frame_expected) {    /* this is what we have been waiting for. */
               to_network_layer(&r.info);   /* pass the data to the network layer */
               inc(frame_expected);         /* next time expect the other sequence nr */
          }
          s.ack = 1 – frame_expected;       /* tell which frame is being acked */
          to_physical_layer(&s);            /* send acknowledgement */
     }
 }
}
```

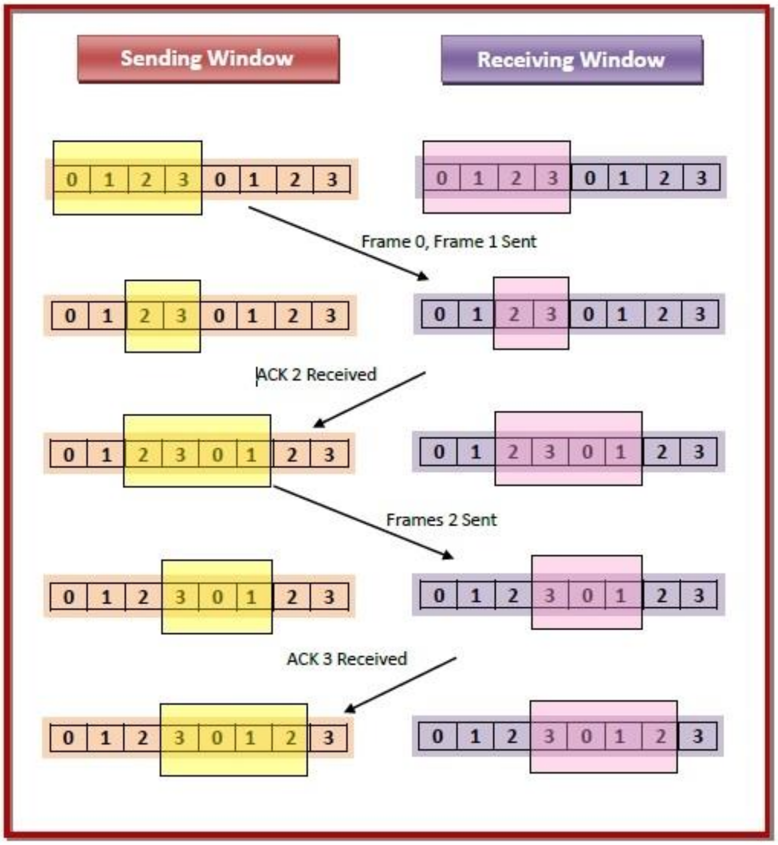
Sliding Window Protocol:

Sliding window protocols are data link layer protocols for reliable and sequential delivery of data frames. The sliding window is also used in Transmission Control Protocol.

In this protocol, multiple frames can be sent by a sender at a time before receiving an acknowledgment from the receiver. The term sliding window refers to the imaginary boxes to hold frames. Sliding window method is also known as windowing.

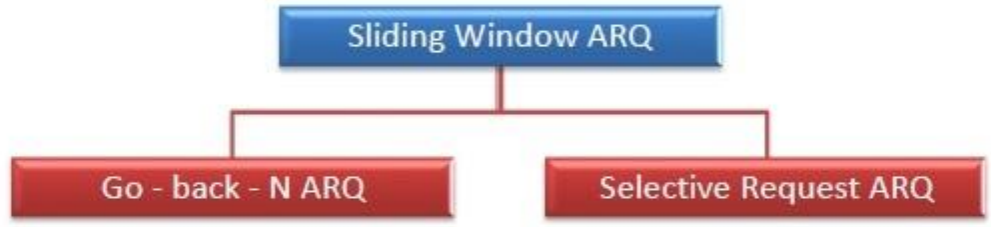
Example:

Suppose that we have sender window and receiver window each of size 4. So, the sequence numbering of both the windows will be 0,1,2,3,0,1,2 and so on. The following diagram shows the positions of the windows after sending the frames and receiving acknowledgments.



Types of Sliding Window Protocols

The Sliding Window ARQ (Automatic Repeat re-quest) protocols are of two categories −



1) **Go-Back-N ARQ:**

Go-Back-N ARQ protocol is also known as Go-Back-N Automatic Repeat Request. It is a data link layer protocol that uses a sliding window method. In this, if any frame is corrupted or lost, all subsequent frames have to be sent again.

The size of the sender window is N in this protocol. Example, Go-Back-8, the size of the sender window, will be 8. The receiver window size is always 1.

If the receiver receives a corrupted frame, it cancels it. The receiver does not accept a corrupted frame. When the timer expires, the sender sends the correct frame again.

2) **Selective Repeat ARQ:**

Selective Repeat ARQ is also known as the Selective Repeat Automatic Repeat Request. It is a data link layer protocol that uses a sliding window method.

The Go-back-N ARQ protocol works well if it has fewer errors. But if there is a lot of error in the frame, lots of bandwidth loss in sending the frames again. So, we use the Selective Repeat ARQ protocol.

In this protocol, the size of the sender window is always equal to the size of the receiver window. The size of the sliding window is always greater than 1.

If the receiver receives a corrupt frame, it does not directly discard it. It sends a negative acknowledgment to the sender.

The sender sends that frame again as soon as on the receiving negative acknowledgment. There is no waiting for any time-out to send that frame.

****