

DATA COMMUNICATION & NETWORKING LAB

PAPER CODE (ETEC-358)

Faculty Name : Ms. Vasudha Bahl

Student name: Sarthak Bhatia

Roll No. : 07714803118

Semester : 6th

Group: I-4



Maharaja Agrasen Institute of Technology, PSP Area, Sector – 22,
Rohini, New Delhi –110086

**DATA COMMUNICATION & NETWORKING
LAB**

PRACTICAL RECORD

PAPER CODE : ETEC-358

Name of the student : Sarthak Bhatia

University Roll No. : 014803118

Branch : IT

Section/ Group : 6I – 4

PRACTICAL DETAILS

Exp. No	Experiment Name	Date of performance	Date of checking	Remarks	Marks(10)
1	Introduction to networking and inter-connecting devices and topology.	05/04/21	08/04/21		
2	PC to PC communication, Parallel Communication using 8-bit parallel cable, Serial Communication using RS 232C. Study of cross and straight cable.	08/04/21	15/04/21		
3	Implement Bit Shuffling.	15/04/21	22/04/21		
4	Implement Character Stuffing or Byte Stuffing.	22/04/21	29/04/21		
5	Implement Cyclic Redundancy Check (CRC).	29/04/21	13/05/21		
6	Implement and study of Stop and Wait Protocol.	13/05/21	20/05/21		
	Implementation and study of Go back-N and Selective	20/05/21	27/05/21		

7	repeat Protocols.				
8	Find shortest path between two nodes in a computer network using Dijkstra's shortest path algorithm.	27/05/21	03/06/21		
9	Implementation of distance vector routing algorithm.	03/06/21	10/03/21		
10	Implementation of Link State routing algorithm.	10/06/21	17/03/21		
11	Write a program for congestion control using Leaky bucket algorithm.	17/06/21	24/03/21		
12	Implementation of Data encryption and decryption (Program for Caesar Cipher and Generalized Caesar Cipher).	24/06/21	01/07/21		

Experiment-1

Date-April 5th,2021

AIM-Introduction to networking and inter-connecting devices and topology.

Network Devices

Hardware devices that are used to connect computers, printers, fax machines and other electronic devices to a network are called network devices. These devices transfer data in a fast, secure and correct way over same or different networks. Network devices may be inter-network or intra- network. Some devices are installed on the device, like RJ45 connector, whereas some are part of the network, like router, switch, etc.

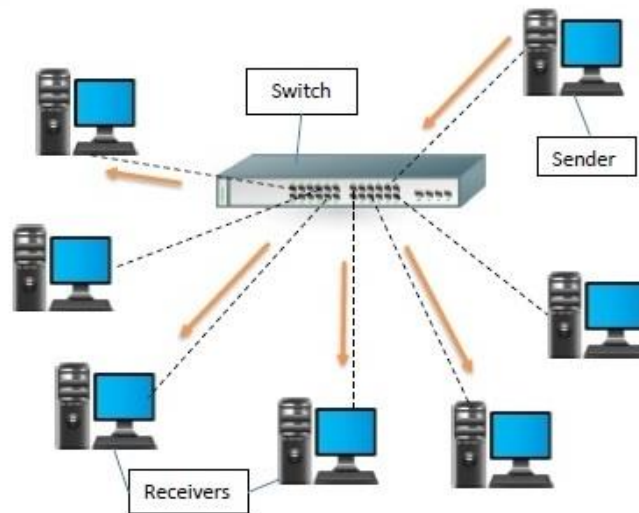
HUB

A hub is a physical layer networking device which is used to connect multiple devices in a network. They are generally used to connect computers in a LAN. A hub has many ports in it. A computer which intends to be connected to the network is plugged in to one of these ports. When a data frame arrives at a port, it is broadcast to every other port, without considering whether it is destined for a particular destination or not.



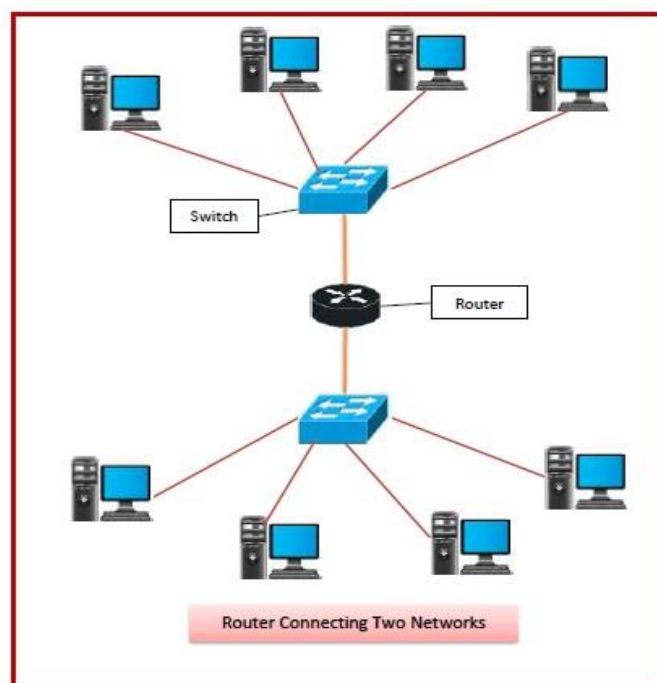
SWITCH

A switch is a data link layer networking device which connects devices in a network and uses packet switching to send and receive data over the network. Like a hub, a switch also has many ports, to which computers are plugged in. However, when a data frame arrives at any port of a network switch, it examines the destination address and sends the frame to the corresponding device(s). Thus, it supports both unicast and multicast communications.



ROUTER

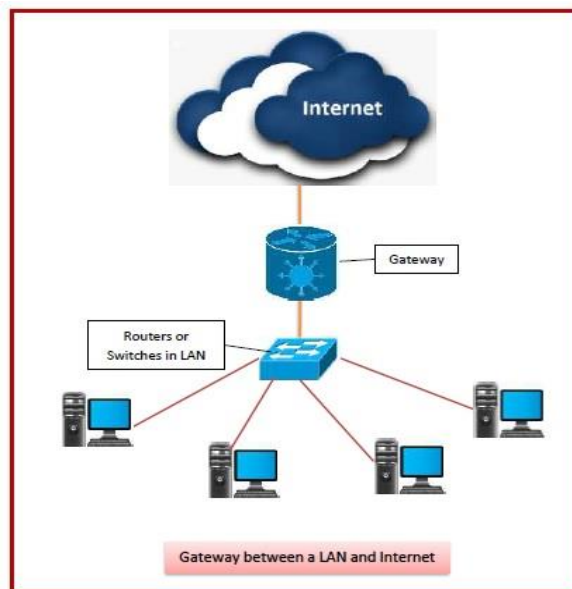
Routers are networking devices operating at layer 3 or a network layer of the OSI model. They are responsible for receiving, analyzing, and forwarding data packets among the connected computer networks. When a data packet arrives, the router inspects the destination address, consults its routing tables to decide the optimal route and then transfers the packet along this route. It connects different networks together and sends data packets from one network to another. A router can be used both in LANs (Local Area Networks) and WANs (Wide Area Networks).



GATEWAY

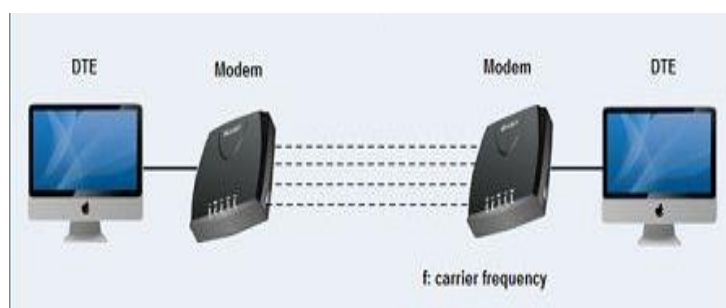
Gateway is a network device used to connect two or more dissimilar networks. Gateway uses packet switching technique to transmit data from one network to

another. A gateway usually is a computer with multiple NICs connected to different networks. A gateway can also be configured completely using software. As networks connect to a different network through gateways, these gateways are usually hosts or end points of the Network.



MODEM

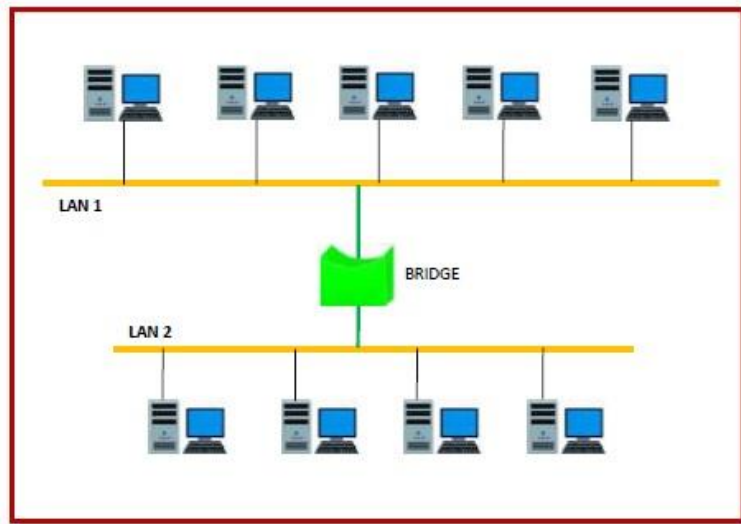
Modem is a device that enables a computer to send or receive data over telephone or cable lines. The function of the modem is to convert digital signal into analog and vice versa. Modem is a combination of two devices, modulator and demodulator. The modulator converts digital data into analog data when the data is being sent by the computer. The demodulator converts analog data signals into digital data when it is being received by the computer.



BRIDGE

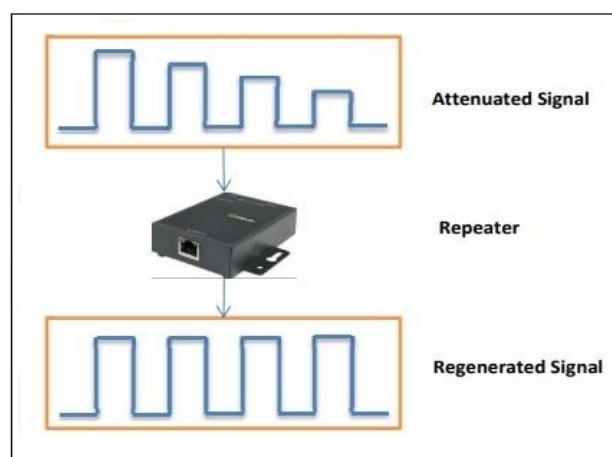
A bridge is a network device that connects multiple LANs (local area networks) together to form a larger LAN. The process of aggregating networks is called

network bridging. A bridge connects the different components so that they appear as parts of a single network. Bridges operate at the data link layer of the OSI model and hence also referred as Layer 2 switches. Bridges connects two or more different LANs that has a similar protocol and provides communication between the devices (nodes) in them.



REPEATER

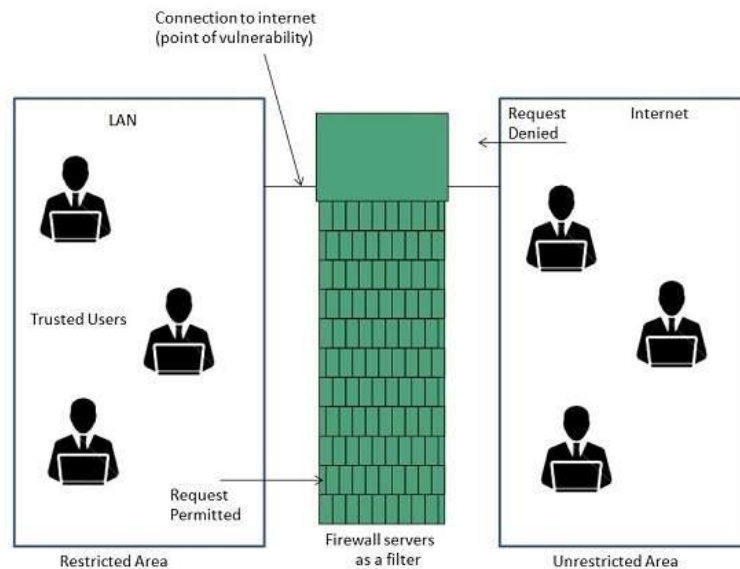
Repeaters are network devices operating at physical layer of the OSI model that amplify or regenerate an incoming signal before re-transmitting it. They are incorporated in networks to expand its coverage area. They are also known as signal boosters. Repeaters amplifies the attenuated signal and then re-transmits it. Digital repeaters can even reconstruct signals distorted by transmission loss. So, repeaters are popularly incorporated to connect between two LANs thus forming a large single LAN.



FIREWALL

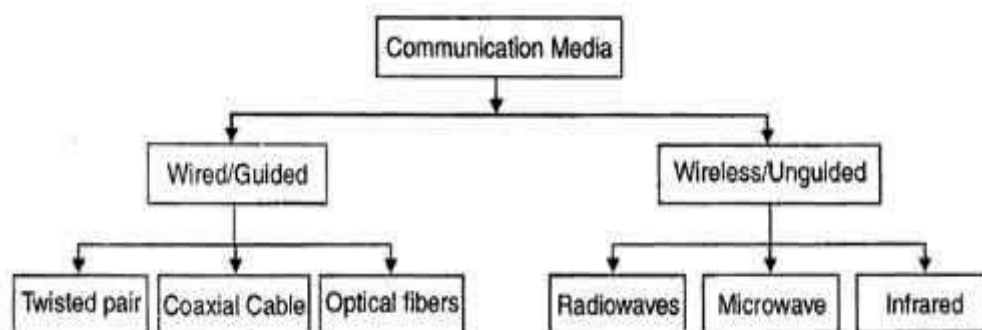
A firewall can be defined as a special type of network security device or a software program that monitors and filters incoming and outgoing network traffic based

on a defined set of security rules. It acts as a barrier between internal private networks and external sources (such as the public Internet). The primary purpose of a firewall is to allow non-threatening traffic and prevent malicious or unwanted data traffic for protecting the computer from viruses and attacks. A firewall is a cybersecurity tool that filters network traffic and helps users block malicious software from accessing the Internet in infected computers.



Transmission media

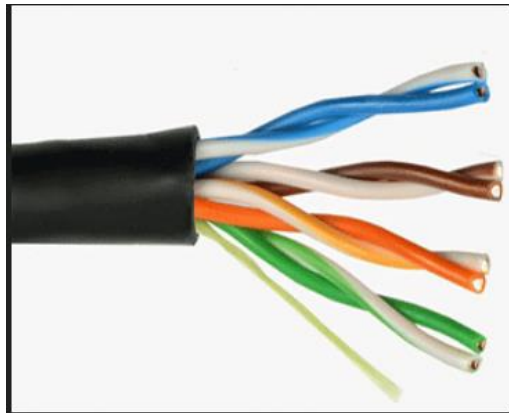
Transmission media is a communication channel that carries the information from the sender to the receiver. Data is transmitted through the electromagnetic signals. The main functionality of the transmission media is to carry the information in the form of bits through LAN. It is a physical path between transmitter and receiver in data communication. The characteristics and quality of data transmission are determined by the characteristics of medium and signal.



Unshielded twisted pair

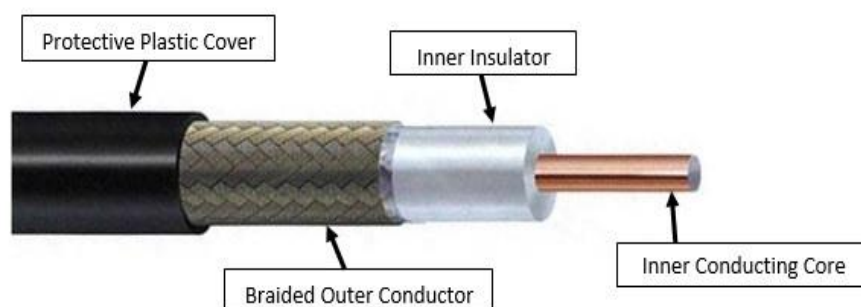
Unshielded twisted pair (UTP) is a ubiquitous type of copper cabling used in telephone wiring and local area networks (LANs). This type of cable has the ability

to block interference and does not depend on a physical shield for this purpose. There are five types of UTP cables -- identified with the prefix CAT, as in *category* - each supporting a different amount of bandwidth. Most enterprises favor UTP cable due to its low cost and ease of installation.



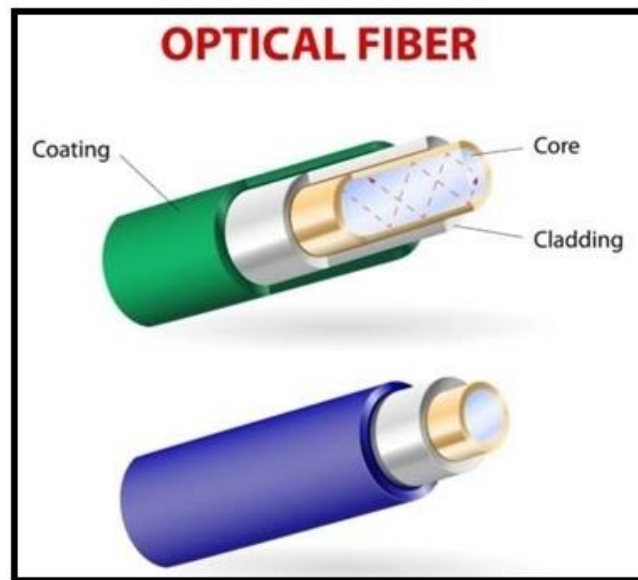
Coaxial cables

Coaxial cables, commonly called coax, are copper cables with metal shielding designed to provide immunity against noise and greater bandwidth. Coax can transmit signals over larger distances at a higher speed as compared to twisted pair cables. Coax has a central core of stiff copper conductor for transmitting signals. This is covered by an insulating material. The insulator is encased by a closely woven braided metal outer conductor that acts as a shield against noise. The outer conductor is again enclosed by a plastic insulating cover.



Fiber Optics

In fiber optic communication, data is transmitted from the source to the destination by sending light pulses through optical fibers. It changes electrical pulses to light signals and vice versa for communication. Fiber optic communications are preferred when a huge amount of data needs to be transmitted across large distances. Optical fiber cables are transparent, flexible fibers made up of glass or plastic through which light waves can pass by the process of total internal reflection.

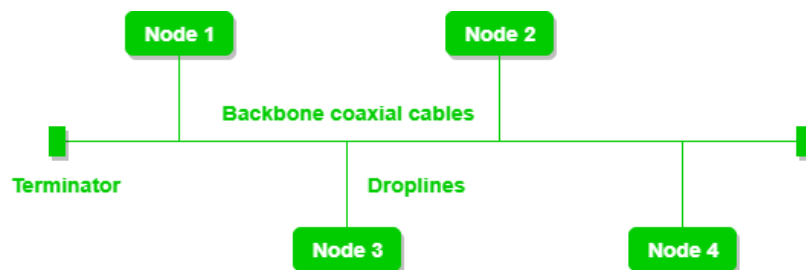


Topologies

Topology defines the structure of the network of how all the components are interconnected to each other. There are two types of topology : physical and logical topology.

Bus Topology

Bus topology is a network type in which every computer and network device is connected to single cable. It transmits the data from one end to another in single direction. No bi-directional feature is in bus topology. It is multi-point connection and a non-robust topology because if the backbone fails the topology crashes.



Advantages:

- If N devices are connected to each other in bus topology, then the number of cables required to connect them is 1 which is known as backbone cable and N drop lines are required.
- Cost of the cable is less as compared to other topology, but it is used to build small networks.

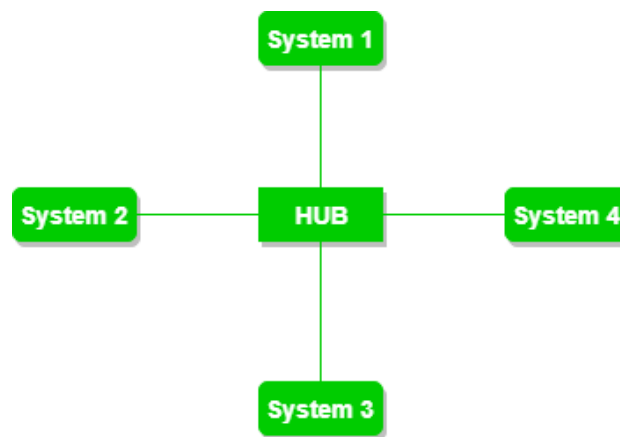
Disadvantages:

- If the common cable fails, then the whole system will crash down.

- If the network traffic is heavy, it increases collisions in the network. To avoid this, various protocols are used in MAC layer.

Star Topology

In star topology, all the devices are connected to a single hub through a cable. This hub is the central node and all other nodes are connected to the central node. The hub can be passive in nature i.e. not intelligent hub such as broadcasting devices, at the same time the hub can be intelligent known as active hubs. Active hubs have repeaters in them.



Advantages:

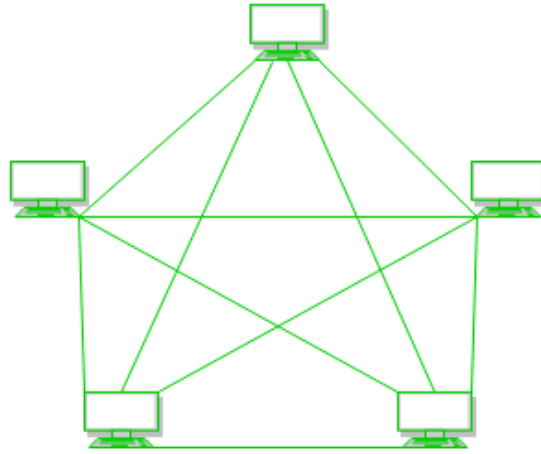
- If N devices are connected to each other in star topology, then the number of cables required to connect them is N. So, it is easy to set up.
- Each device requires only 1 port i.e. to connect to the hub.

Disadvantages:

- If the concentrator (hub) on which the whole topology relies fails, the whole system will crash down.
- Cost of installation is high.
- Performance is based on the single concentrator i.e. hub.

Mesh Topology

In mesh topology, every device is connected to another device via particular channel. If N number of devices are connected with each other in mesh topology, then total number of ports that is required by each device is N-1.



Advantages :

- It is robust.
- Fault is diagnosed easily. Data is reliable because data is transferred among the devices through dedicated channels or links.
- Provides security and privacy.

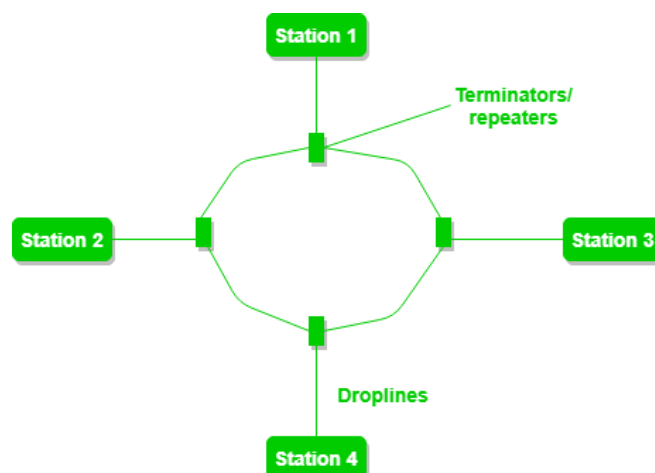
Disadvantages :

- Installation and configuration is difficult.
- Cost of cables are high as bulk wiring is required, hence suitable for less number of devices.
- Cost of maintenance is high.

Ring Topology

In this topology, it forms a ring connecting devices with its exactly two neighboring devices. A number of repeaters are used for Ring topology with a large number of nodes, because if someone wants to send some data to the last node in the ring topology with 100 nodes, then the data will have to pass through 99 nodes to reach the 100th node. Hence to prevent data loss repeaters are used in the network.

The transmission is unidirectional, but it can be made bidirectional by having 2 connections between each Network Node, it is called Dual Ring Topology.



Advantages:

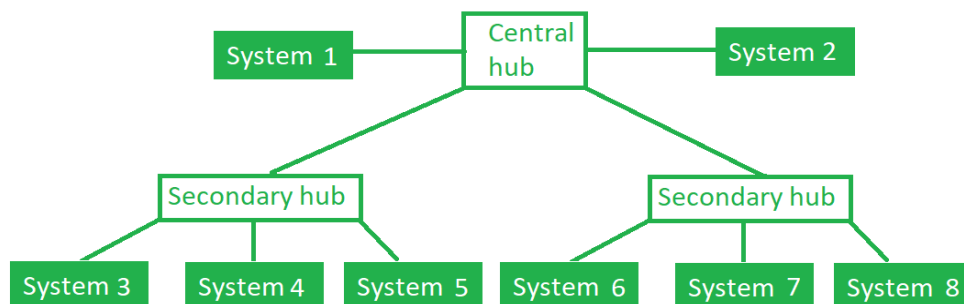
- The possibility of collision is minimum in this type of topology.
- Cheap to install and expand.

Disadvantages:

- Troubleshooting is difficult in this topology.
- Addition of stations in between or removal of stations can disturb the whole topology.

Tree Topology

This topology is the variation of Star topology. This topology have hierarchical flow of data. In this the various secondary hubs are connected to the central hub which contains the repeater. In this data flow from top to bottom i.e. from the central hub to secondary and then to the devices or from bottom to top i.e. devices to secondary hub and then to the central hub. It is multi-point connection and a non-robust topology because if the backbone fails the topology crashes.

**Advantages:**

- It allows more devices to be attached to a single central hub thus it increases the distance that is travel by the signal to come to the devices.
- It allows the network to get isolate and also prioritize from different computers.

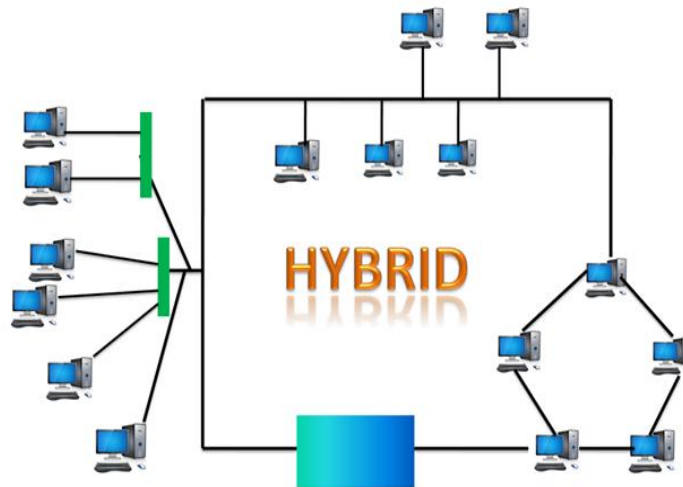
Disadvantages :

- If the central hub gets fails the entire system fails.
- The cost is high because of cabling.

Hybrid Network

The combination of various different topologies is known as Hybrid topology. A Hybrid topology is a connection between different links and nodes to transfer the

data. When two or more different topologies are combined together is termed as Hybrid topology and if similar topologies are connected with each other will not result in Hybrid topology.



Advantages :

- Failure of any part of the network will not affect the functioning of rest of the network.
- Size of the network can be easily expanded by adding new devices without affecting the functionality of the existing network.
- It is very flexible as it can be designed according to the requirements of the organization.
- Hybrid topology is very effective as it can be designed in such a way that the strength of the network is maximized and weakness of the network is minimized.

Disadvantages:

- The major drawback of the Hybrid topology is the design of the Hybrid network. It is very difficult to design the architecture of the Hybrid network.
- The Hubs used in the Hybrid topology are very expensive as these hubs are different from usual Hubs used in other topologies.
- The infrastructure cost is very high as a hybrid network requires a lot of cabling, network devices, etc.

Experiment-2

Date-April 8th,2021

AIM- PC to PC communication , Parallel Communication using 8 bit parallel cable , Serial Communication using RS 232C. Study of cross and straight cable. Study of various networking commands.

Unshielded Twisted Pair(UTP) Cable

One of the earliest guided transmission media is twisted pair cables. A twisted pair cable comprises of two separate insulated copper wires, which are twisted together and run in parallel. The copper wires are typically 1mm in diameter. One of the wires is used to transmit data and the other is the ground reference. There are two types of twisted pair cables –

- Unshielded Twisted Pair (UTP): These generally comprise of wires and insulators.
- Shielded Twisted Pair (STP): They have a braided wired mesh that encases each pair of insulated wires.

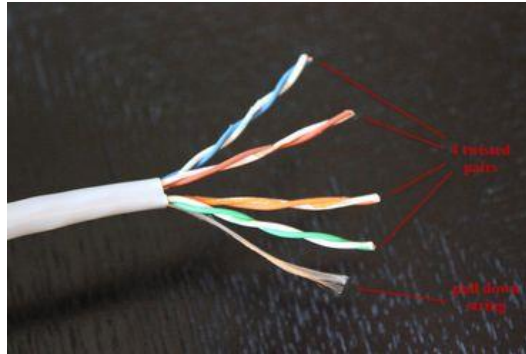
The cable can be categorized as Cat 5, Cat 5e, Cat 6 UTP cable. Cat 5 UTP cable can support 10/100 Mbps Ethernet network, whereas Cat 5e and Cat 6 UTP cable can support Ethernet network running at 10/100/1000 Mbps. You might heard about Cat 3 UTP cable, it's not popular anymore since it can only support 10 Mbps Ethernet network. Straight and crossover cable can be Cat3, Cat 5, Cat 5e or Cat 6 UTP cable, the only difference is each type will have different wire arrangement in the cable for serving different purposes.

UTP cables are often groups of twisted pairs grouped together with colour coded insulators, the number of which depends on the purpose. All transmissions are prone to noise, interferences, and crosstalk. When the wires are twisted, some part of the noise signals is in the direction of data signals while the other parts are in the opposite directions. Thus the external waves cancel out due to the different twists. The receiver calculates the difference in the voltages of the two wires for retrieving data. Thus a much better immunity against noise is obtained.

Construction of Straight and Cross Cable

1: Strip the cable jacket about 1.5 inch down from the end.

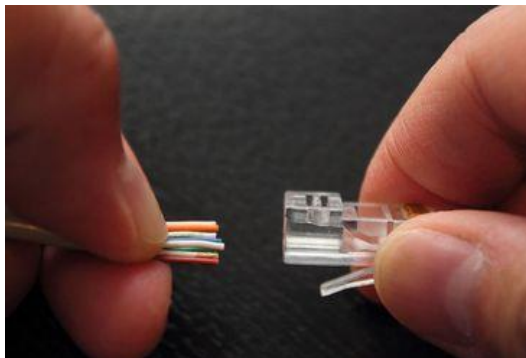
2: Spread the four pairs of twisted wire apart. For Cat 5e, you can use the pull string to strip the jacket farther down if you need to, then cut the pull string. Cat 6 cables have a spine that will also need to be cut.



3: Untwist the wire pairs and neatly align them in the T568B orientation. Be sure not to untwist them any farther down the cable than where the jacket begins; we want to leave as much of the cable twisted as possible.

4: Cut the wires as straight as possible, about 0.5 inch above the end of the jacket.

5: Carefully insert the wires all the way into the modular connector, making sure that each wire passes through the appropriate guides inside the connector.

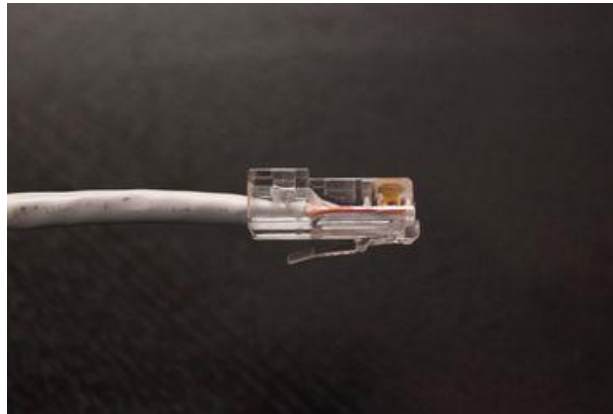


6: Push the connector inside the crimping tool and squeeze the crimper all the way down.



7: Repeat steps 1-6 for the other end of the cable.









8: To make sure you've successfully terminated each end of the cable, use a cable tester to test each pin.



For crossover cables, simply make one end of the cable a T568A and the other end a T568B.

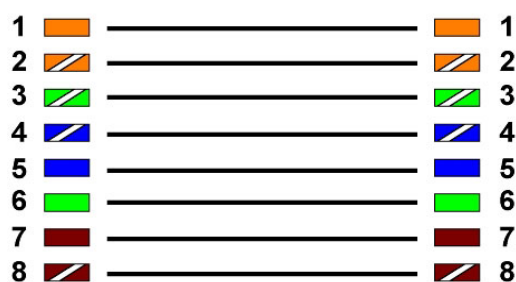
Colour Codes

Horizontal UTP cable is four-pair construction by industry cabling standard. Each pair has two conductors. One wire of the pair is assigned the pair colour with a white stripe and the other wire is assigned the colour white with the pair colour stripe. The table below lists the pair and colour code for a four-pair horizontal UTP cable.

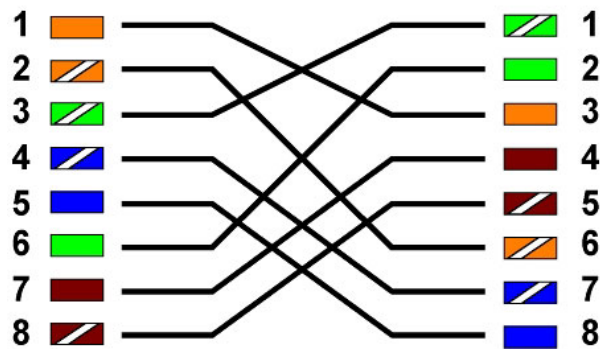
Wire Number	Pair Number	Colour
1	1	 white/blue
2	1	 blue
3	2	 white/orange
4	2	 orange
5	3	 white/green
6	3	 green
7	4	 white/brown
8	4	 brown

Pin Configuration Diagram

EIA/TIA T568B Straight Through Diagram



EIA/TIA T568B Crossover Diagram



Commands

- 1) **Ipconfig** - ipconfig is a console application designed to run from the Windows command prompt. This utility allows you to get the IP address information of a Windows computer. It also allows some control over your network adapters, IP addresses.

Syntax : ipconfig;

```
Command Prompt
C:\Users\>ipconfig

Windows IP Configuration

Ethernet adapter vEthernet (Wide Networking Switch (jg)):

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::4099:
    IPv4 Address. . . . . : 10.0.0.2
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 10.0.0.1

Wireless LAN adapter Wi-Fi:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 

Wireless LAN adapter Local Area Connection* 1:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 

Wireless LAN adapter Local Area Connection* 3:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 

Ethernet adapter vEthernet (Default Switch):
```

- 2) **Ping** – ping is the primary TCP/IP command used to troubleshoot connectivity, reachability, and name resolution. The ping command sends packets of data to a specific IP address on a network, and then lets you know how long it took to transmit that data and get a response. It's a handy tool that you can use to quickly test various points of your network.

Syntax : ping url;

```
C:\Users\sarth>ping google.com

Pinging google.com [172.217.166.14] with 32 bytes of data:
Reply from 172.217.166.14: bytes=32 time=11ms TTL=116
Reply from 172.217.166.14: bytes=32 time=11ms TTL=116
Reply from 172.217.166.14: bytes=32 time=11ms TTL=116
Reply from 172.217.166.14: bytes=32 time=11ms TTL=116

Ping statistics for 172.217.166.14:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 11ms, Maximum = 11ms, Average = 11ms
```

- 3) **Netstat** – The network statistics (`netstat`) command is a networking tool used for troubleshooting and configuration, that can also serve as a monitoring tool for connections over the network. Both incoming and outgoing connections, routing tables, port listening, and usage statistics are common uses for this command

Syntax : netstat;

```
C:\Users\sarth>netstat

Active Connections

Proto Local Address           Foreign Address         State
TCP    127.0.0.1:50465          LAPTOP-CK054HLR:50466  ESTABLISHED
TCP    127.0.0.1:50466          LAPTOP-CK054HLR:50465  ESTABLISHED
TCP    127.0.0.1:54147          LAPTOP-CK054HLR:54148  ESTABLISHED
TCP    127.0.0.1:54148          LAPTOP-CK054HLR:54147  ESTABLISHED
TCP    127.0.0.1:54149          LAPTOP-CK054HLR:54150  ESTABLISHED
TCP    127.0.0.1:54150          LAPTOP-CK054HLR:54149  ESTABLISHED
TCP    127.0.0.1:54151          LAPTOP-CK054HLR:54152  ESTABLISHED
TCP    127.0.0.1:54152          LAPTOP-CK054HLR:54151  ESTABLISHED
TCP    127.0.0.1:54153          LAPTOP-CK054HLR:54154  ESTABLISHED
TCP    127.0.0.1:54154          LAPTOP-CK054HLR:54153  ESTABLISHED
TCP    127.0.0.1:63650          LAPTOP-CK054HLR:63651  ESTABLISHED
TCP    127.0.0.1:63651          LAPTOP-CK054HLR:63650  ESTABLISHED
TCP    127.0.0.1:63937          LAPTOP-CK054HLR:63938  ESTABLISHED
TCP    127.0.0.1:63938          LAPTOP-CK054HLR:63937  ESTABLISHED
TCP    127.0.0.1:63950          LAPTOP-CK054HLR:63951  ESTABLISHED
TCP    127.0.0.1:63951          LAPTOP-CK054HLR:63950  ESTABLISHED
```

- 4) **Tracert** - The `tracert` command (spelled `traceroute` in Unix/Linux implementations) is one of the key diagnostic tools for TCP/IP. It displays a list of all the routers that a packet must go through to get from the computer where `tracert` is run to any other computer on the Internet.

Syntax : tracert /?;

```

C:\Users\sarth>tracert /?

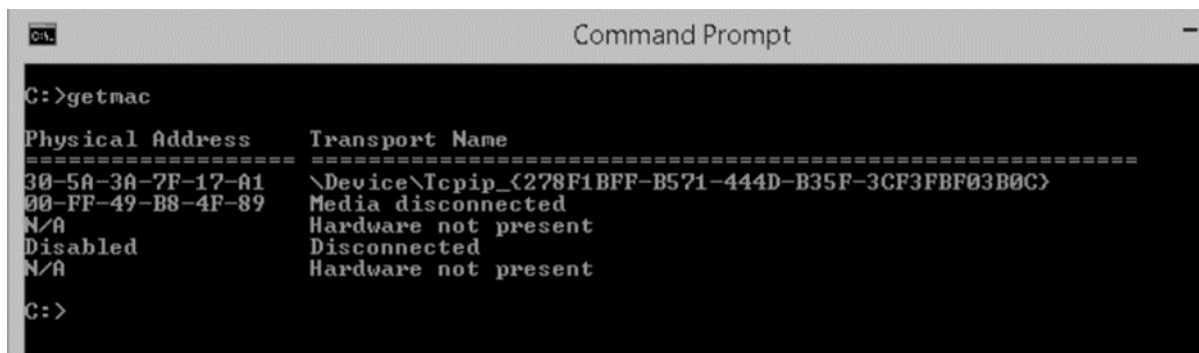
Usage: tracert [-d] [-h maximum_hops] [-j host-list] [-w timeout]
              [-R] [-S srcaddr] [-4] [-6] target_name

Options:
    -d                Do not resolve addresses to hostnames.
    -h maximum_hops   Maximum number of hops to search for target.
    -j host-list       Loose source route along host-list (IPv4-only).
    -w timeout         Wait timeout milliseconds for each reply.
    -R                Trace round-trip path (IPv6-only).
    -S srcaddr         Source address to use (IPv6-only).
    -4                Force using IPv4.
    -6                Force using IPv6.

```

- 5) **Getmac** - Returns the media access control (MAC) address and list of network protocols associated with each address for all network cards in each computer, either locally or across a network. This command is particularly useful either when you want to enter the MAC address into a network analyser, or when you need to know what protocols are currently in use on each network adapter on a computer.

Syntax : getmac;



```

C:\>getmac

Physical Address      Transport Name
=====
30-5A-3A-7F-17-A1     \Device\NPF{278F1BFF-B571-444D-B35F-3CF3F8F03B0C}
00-FF-49-B8-4F-89     Media disconnected
N/A                   Hardware not present
Disabled              Disconnected
N/A                   Hardware not present

C:\>

```

Experiment-3

Date- April 15th,2021

AIM- Implement Bit Shuffling.

Bit Shuffling

Data link layer is responsible for something called Framing, which is the division of stream of bits from network layer into manageable units (called frames). Frames could be of fixed size or variable size. In variable-size framing, we need a way to define the end of the frame and the beginning of the next frame.

Bit stuffing is the insertion of non information bits into data. Note that stuffed bits should not be confused with overhead bits. **Overhead bits** are non-data bits that are necessary for transmission (usually as part of headers, check sums etc.).

Example of bit stuffing –

Bit sequence: 110101111101011111101011111110 (without bit stuffing)

Bit sequence: 1101011111**00**1011111**0**10101111**0**110 (with bit stuffing)

Purpose of Bit Stuffing

In Data Link layer, the stream of bits from the physical layer is divided into data frames. The data frames can be of fixed length or variable length. In variable - length framing, the size of each frame to be transmitted may be different. So, a pattern of bits is used as a delimiter to mark the end of one frame and the beginning of the next frame. However, if the pattern occurs in the message, then mechanisms needs to be incorporated so that this situation is avoided.

The two common approaches are –

Byte - Stuffing – A byte is stuffed in the message to differentiate from the delimiter. This is also called character-oriented framing.

Bit - Stuffing – A pattern of bits of arbitrary length is stuffed in the message to differentiate from the delimiter. This is also called bit - oriented framing.

Frame in a Bit - Oriented Protocol

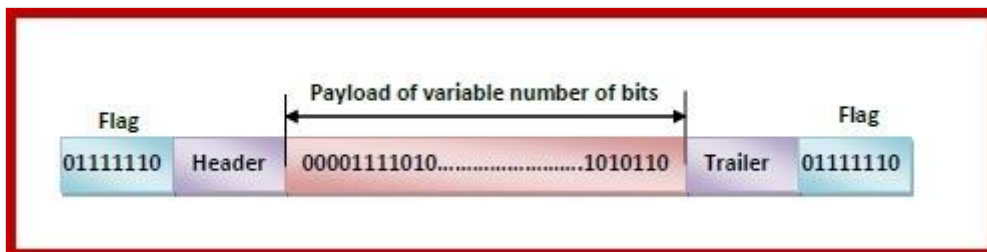
In bit-oriented protocols, the message is coded as a sequence of bits, which are interpreted in the upper layers as text, graphics, audio, video etc. A frame has the following parts –

Frame Header – It contains the source and the destination addresses of the frame.

Payload field – It contains the message to be delivered.

Trailer – It contains the error detection and error correction bits.

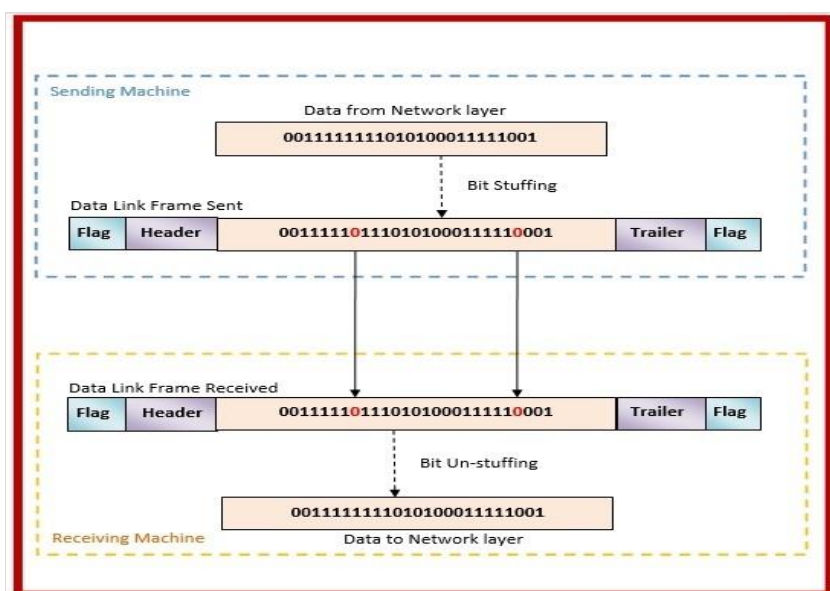
Flags – A bit pattern that defines the beginning and end bits in a frame. It is generally of 8-bits. Most protocols use the 8-bit pattern 01111110 as flag.



Bit Stuffing Mechanism

In a data link frame, the delimiting flag sequence generally contains six or more consecutive 1s. In order to differentiate the message from the flag in case of the same sequence, a single bit is stuffed in the message. Whenever a 0 bit is followed by five consecutive 1bits in the message, an extra 0 bit is stuffed at the end of the five 1s.

When the receiver receives the message, it removes the stuffed 0s after each sequence of five 1s. The un-stuffed message is then sent to the upper layers.



Program (Java)

```
import java.util.*;

public class bitStuffing{

    public static void main(String[] args) {

        Scanner scn=new Scanner(System.in);

        System.out.println("Enter input bits:- ");

        String str=scn.next();

        String ans="";

        int i=0;

        while(i<str.length()){

            char ch=str.charAt(i);

            if(ch=='1'){

                ans+=ch;

                int j=i+1;

                int count=1;

                while(j<str.length() && str.charAt(j)=='1' && count<5){

                    count++;

                    ans+=str.charAt(j);

                    j++;

                }

                if(count==5){

                    ans+='0';

                }

                i=j-1;

            }else{

                ans+=ch;

            }

        }

    }

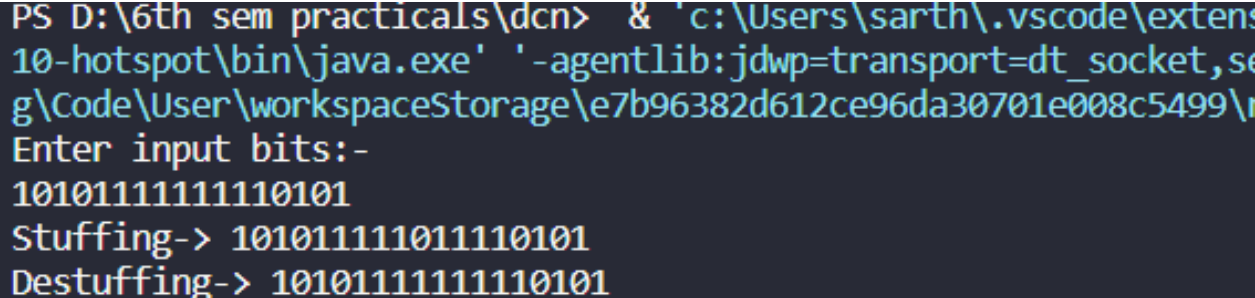
}
```

```
    }  
    i++;  
}  
System.out.println("Stuffing-> "+ans);  
String deStuffing="";  
i=0;  
while(i<ans.length()){  
    char ch=ans.charAt(i);  
    if(ch=='1'){  
        deStuffing+=ch;  
        int j=i+1;  
        int count=1;  
        while(j<ans.length() && ans.charAt(j)=='1' && count<5){  
            count++;  
            deStuffing+=ans.charAt(j);  
            j++;  
        }  
        i=j-1;  
        if(count==5){  
            i++;  
        }  
    }else{  
        deStuffing+=ch;  
    }  
    i++;  
}
```



```
System.out.println("Destuffing-> "+deStuffing);  
scn.close();  
}  
}
```

Output



```
PS D:\6th sem practicals\dcn> & 'c:\Users\sarth\.vscode\extensions\ms-vscode.java-10-hotspot\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,se  
g\Code\User\workspaceStorage\e7b96382d612ce96da30701e008c5499\p  
Enter input bits:-  
1010111111110101  
Stuffing-> 10101111011110101  
Destuffing-> 1010111111110101
```

Experiment-4

Date-April 22, 2021.

AIM- Implement Character Stuffing or Byte Stuffing.

Byte/Character Stuffing

In Data Link layer, the stream of bits from physical layer are divided into data frames. The data frames can be of fixed length or variable length. In variable – length framing, the size of each frame to be transmitted may be different. So, a pattern of bits is used as a delimiter to mark the end of one frame and the beginning of the next frame. However, if the pattern occurs in the message, then mechanisms needs to be incorporated so that this situation is avoided.

The two common approaches are –

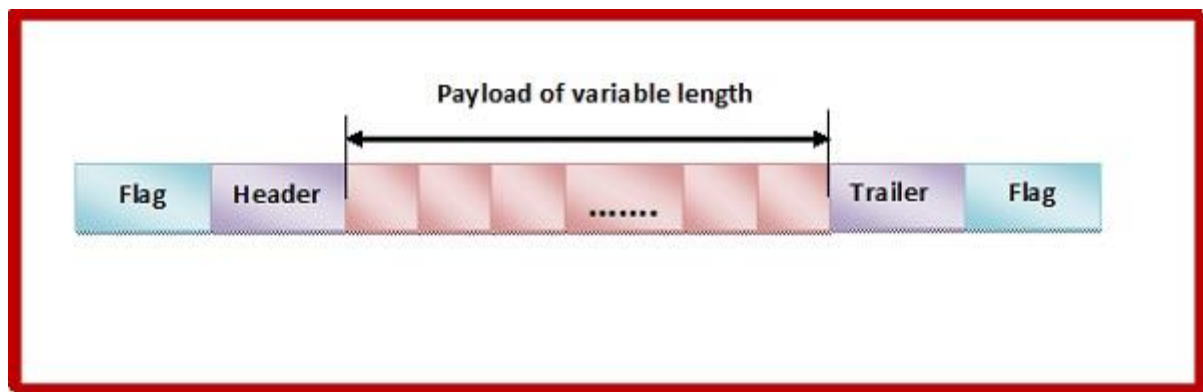
- **Byte – Stuffing** – A byte is stuffed in the message to differentiate from the delimiter. This is also called character-oriented framing.
- **Bit – Stuffing** – A pattern of bits of arbitrary length is stuffed in the message to differentiate from the delimiter. This is also called bit – oriented framing.

Frame in a Character – Oriented Framing

In character – oriented protocols, the message is coded as 8-bit characters, using codes like ASCII codes.

A frame has the following parts –

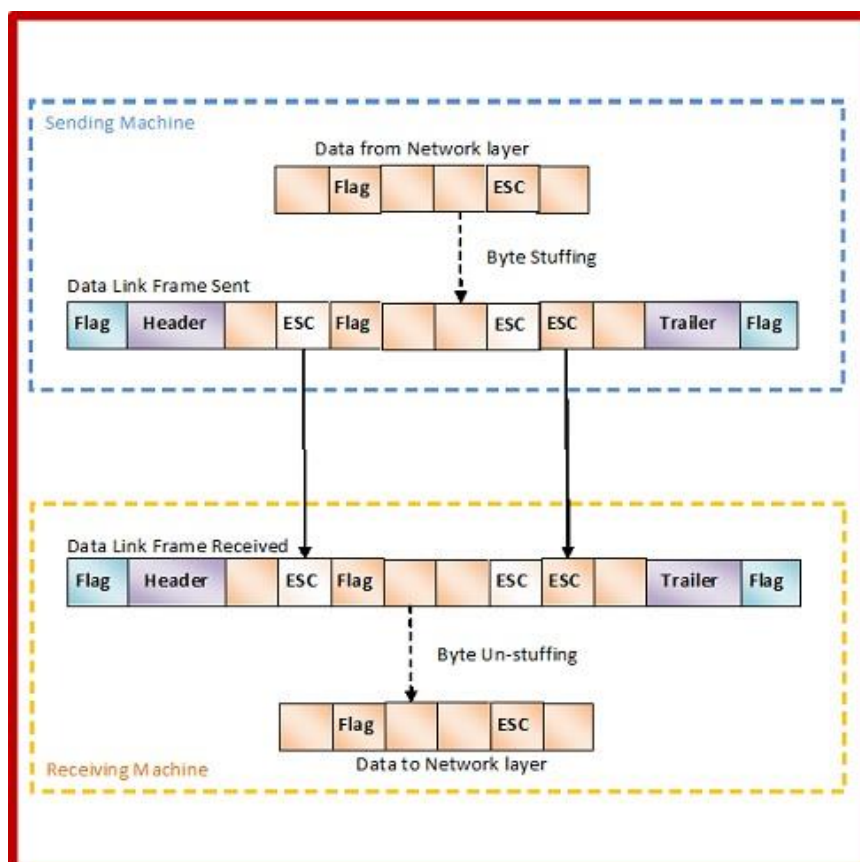
- **Frame Header** – It contains the source and the destination addresses of the frame.
- **Payload field** – It contains the message to be delivered.
- **Trailer** – It contains the error detection and error correction bits.
- **Flags** – 1- byte (8-bits) flag at the beginning and at end of the frame. It is a protocol – dependent special character, signalling the start and end of the frame.



Byte Stuffing Mechanism

If the pattern of the flag byte is present in the message byte, there should be a strategy so that the receiver does not consider the pattern as the end of the frame. In character – oriented protocol, the mechanism adopted is byte stuffing.

In byte stuffing, a special byte called the escape character (ESC) is stuffed before every byte in the message with the same pattern as the flag byte. If the ESC sequence is found in the message byte, then another ESC byte is stuffed before it.



Program (Java)

```
import java.util.*;

public class byteStuffing {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the Message to be Sent : ");

        String data = sc.nextLine();

        String res = new String();

        data = '$' + data + '$';

        for (int i = 1; i < data.length()-1; i++) {

            if (data.charAt(i) == '$')

                res = res + '#' + data.charAt(i);

            else if (data.charAt(i) == '#')

                res = res + '#' + data.charAt(i);

            else

                res = res + data.charAt(i);

        }

        System.out.println();

        System.out.println("The data after byte stuffing : " + res);

        String out = "";

        for (int i = 1; i < res.length() - 1; i++) {

            if (res.charAt(i) == '$' || res.charAt(i) == '#')

                continue;

            else{

                out+=res.charAt(i);

            }

        }

        System.out.println(out);

    }

}
```

```
    }  
}  
System.out.println();  
System.out.println("The data after byte de-stufing : " + out);  
sc.close();  
}  
}
```

Output

```
Enter the Message to be Sent :  
abcdef$ghij#klm  
  
The data after byte stufing : $abcdef#$ghij##klm$  
  
The data after byte de-stufing : abcdefghijklm  
PS D:\6th sem practicals\dcn>
```

Experiment-5

Date-April 29, 2021.

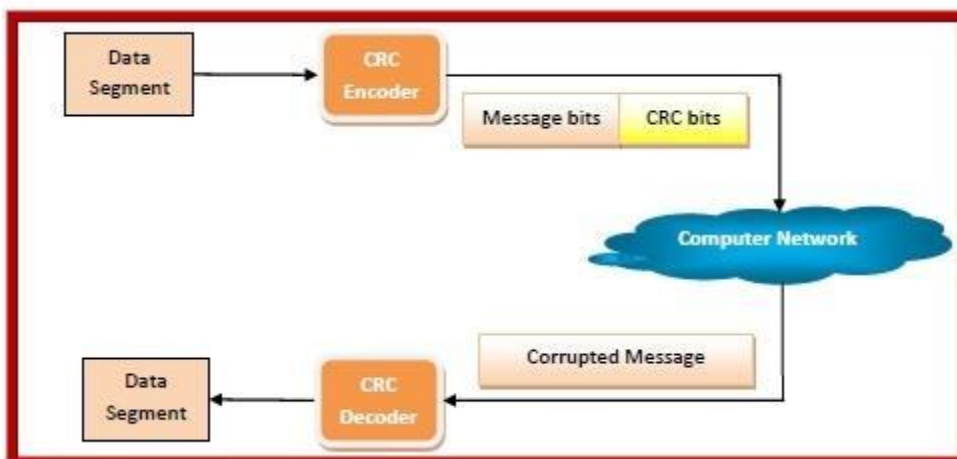
AIM- Implement Cyclic Redundancy Check(CRC).

Cyclic Redundancy Check (CRC)

Cyclic Redundancy Check (CRC) is a block code invented by W. Wesley Peterson in 1961. It is commonly used to detect accidental changes to data transmitted via telecommunications networks and storage devices.

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. So, CRC is also called polynomial code checksum.

The process is illustrated as follows –



Encoding using CRC

- The communicating parties agree upon the size of message block and the CRC divisor. For example, the block chosen may be CRC (7, 4), where 7 is the total length of the block and 4 is the number of bits in the data segment. The divisor chosen may be 1011.
- The sender performs binary division of the data segment by the divisor.
- It then appends the remainder called CRC bits to the end of data segment. This makes the resulting data unit exactly divisible by the divisor.

Decoding

- 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. The receiver may then send an erroneous acknowledgement back to the sender for re-transmission.

Program (Java)

```
import java.util.*;

public class crc {

    public static void main(String args[]) {

        Scanner scan = new Scanner(System.in);

        int n;

        System.out.print("Enter the size of the data : ");

        n = scan.nextInt();

        int data[] = new int[n];

        System.out.print("\nEnter the data : ");

        for(int i=0 ; i < n ; i++) {

            data[i] = scan.nextInt();

        }

        System.out.print("\nEnter the size of the divisor : ");

        n = scan.nextInt();

        int divisor[] = new int[n];

        System.out.print("\nEnter the divisor : ");

        for(int i=0 ; i < n ; i++) {

            divisor[i] = scan.nextInt();

        }

        int remainder[] = divide(data, divisor);

        System.out.print("\nRemainder Generated is : ");
```

```

        for(int i=0 ; i < remainder.length-1 ; i++) {
            System.out.print(remainder[i]);
        }
        int sent_data[] = new int[data.length + remainder.length - 1];
        int j = 0;
        System.out.print("\nThe CRC code generated is :");
        for(int i=0 ; i < data.length ; i++) {
            System.out.print(data[i]);
            sent_data[j] = data[i];
            j++;
        }
        for(int i=0 ; i < remainder.length-1 ; i++) {
            System.out.print(remainder[i]);
            sent_data[j] = remainder[i];
            j++;
        }
        receive(sent_data, divisor);
    scan.close();
}

static int[] divide(int old_data[], int divisor[]) {
    int remainder[] , i;
    int data[] = new int[old_data.length + divisor.length];
    System.arraycopy(old_data, 0, data, 0, old_data.length);
    remainder = new int[divisor.length];
    System.arraycopy(data, 0, remainder, 0, divisor.length);
    for(i=0 ; i < old_data.length ; i++) {

```



```

        if(remainder[0] == 1) {
            for(int j=1 ; j < divisor.length ; j++) {
                remainder[j-1] = exor(remainder[j], divisor[j]);
            }
        }
        else {
            for(int j=1 ; j < divisor.length ; j++) {
                remainder[j-1] = exor(remainder[j], 0);
            }
        }
        remainder[divisor.length-1] = data[i+divisor.length];
    }
    return remainder;
}

static int exor(int a, int b) {
    if(a == b)
        return 0;
    return 1;
}

static void receive(int data[], int divisor[]) {
    int remainder[] = divide(data, divisor);
    int max = 0;
    System.out.print("\nRemainder Generated at reciever end is : ");
    for(int i=0 ; i < remainder.length-1 ; i++) {
        System.out.print(remainder[i]);
        if(remainder[i] > max){

```

```

        max = remainder[i];
    }
}
if(max == 1){
    System.out.println("\n\nThere is an error in received data...");
}
else{
    System.out.println("\n\nData was received without any error.");
}
}
}

```

Output

```

Enter the size of the data : 6

Enter the data : 1
0
0
1
0
0

Enter the size of the divisor : 4

Enter the divisor : 1
1
0
1

Remainder Generated is : 001
The CRC code generated is :100100001
Remainder Generated at reciever end is : 000

Data was received without any error.

```

Experiment-6

Date-May 13, 2021.

AIM- Implement and study of Stop and Wait Protocol.

Stop and Wait Protocol

Before understanding the stop and Wait protocol, we first know about the error control mechanism. The error control mechanism is used so that the received data should be exactly same whatever sender has sent the data. The error control mechanism is divided into two categories, i.e., Stop and Wait ARQ and sliding window. The sliding window is further divided into two categories, i.e., Go Back N, and Selective Repeat. Based on the usage, the people select the error control mechanism whether it is **stop and wait** or **sliding window**.

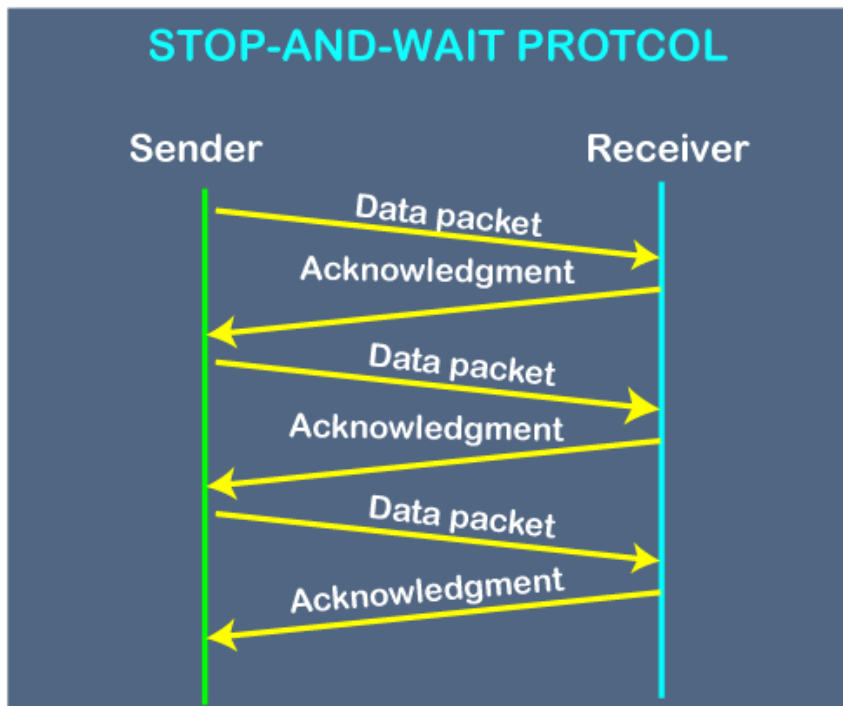
Stop and wait means, whatever the data that sender wants to send, he sends the data to the receiver. After sending the data, he stops and waits until he receives the acknowledgment from the receiver. The stop and wait protocol is a flow control protocol where flow control is one of the services of the data link layer.

It is a data-link layer protocol which is used for transmitting the data over the noiseless channels. It provides unidirectional data transmission which means that either sending or receiving of data will take place at a time. It provides flow-control mechanism but does not provide any error control mechanism.

The idea behind the usage of this frame is that when the sender sends the frame then he waits for the acknowledgment before sending the next frame.

Working of Stop and Wait protocol

The figure below shows the working of the stop and wait protocol. If there is a sender and receiver, then sender sends the packet and that packet is known as a data packet. The sender will not send the second packet without receiving the acknowledgment of the first packet. The receiver sends the acknowledgment for the data packet that it has received. Once the acknowledgment is received, the sender sends the next packet. This process continues until all the packet are not sent. The main advantage of this protocol is its simplicity but it has some disadvantages also. For example, if there are 1000 data packets to be sent, then all the 1000 packets cannot be sent at a time as in Stop and Wait protocol, one packet is sent at a time.



Primitives of Stop and Wait Protocol

The primitives of stop and wait protocol are:

Sender side:

Rule 1: Sender sends one data packet at a time.

Rule 2: Sender sends the next packet only when it receives the acknowledgment of the previous packet.

Therefore, the idea of stop and wait protocol in the sender's side is very simple, i.e., send one packet at a time, and do not send another packet before receiving the acknowledgment.

Receiver side:

Rule 1: Receive and then consume the data packet.

Rule 2: When the data packet is consumed, receiver sends the acknowledgment to the sender.

Therefore, the idea of stop and wait protocol in the receiver's side is also very simple, i.e., consume the packet, and once the packet is consumed, the acknowledgment is sent. This is known as a flow control mechanism.

PROGRAM (C++)

```
#include<iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    cout<<"Enter number of frames : ";
```

```
    int n;
```

```
    cin>>n;
```

```
    int max_time = 5;
```

```
    cout<<"Max Time for receving ack : "<<max_time << endl;
```

```
    int totaltime = 0;
```

```
    int frame_number = 1;
```

```
    while(frame_number <= n){
```

```
        cout<<"\nFrame number : "<<frame_number<<endl;
```

```
        cout<<"Enter time to receive the Ack : ";
```

```
        int time_taken;
```

```
        cin>>time_taken;
```

```
        cout<<endl;
```

```
        if(time_taken <= max_time){
```

```
            cout<<"Acknowledgement Received\n"<<endl;
```

```
            totaltime += time_taken;
```

```
            frame_number++;
```

```
        }
```

```
        else{
```

```
            cout<<"No Acknowledgement Received\n"<<endl;
```

```
            totaltime += time_taken - max_time;
```

```
        }
```

```
    }
```

```
    cout<<"Total time : "<<totaltime;
```

```
    return 0;
```

```
}
```

OUTPUT-

```
Enter number of frames : 3
Max Time for receiving ack : 5

Frame number : 1
Enter time to receive the Ack : 4

Acknowledgement Received

Frame number : 2
Enter time to receive the Ack : 3

Acknowledgement Received

Frame number : 3
Enter time to receive the Ack : 7

No Acknowledgement Received

Frame number : 3
Enter time to receive the Ack : 5

Acknowledgement Received

Total time : 14

...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment-7

Date-May 20, 2021.

AIM- Implementation and study of Go back-N and Selective repeat Protocols.

Go-Back-N Protocol

Go-Back-N protocol, also called Go-Back-N Automatic Repeat request, is a data link layer protocol that uses a sliding window method for reliable and sequential delivery of data frames. It is a case of sliding window protocol having to send window size of N and receiving window size of 1.

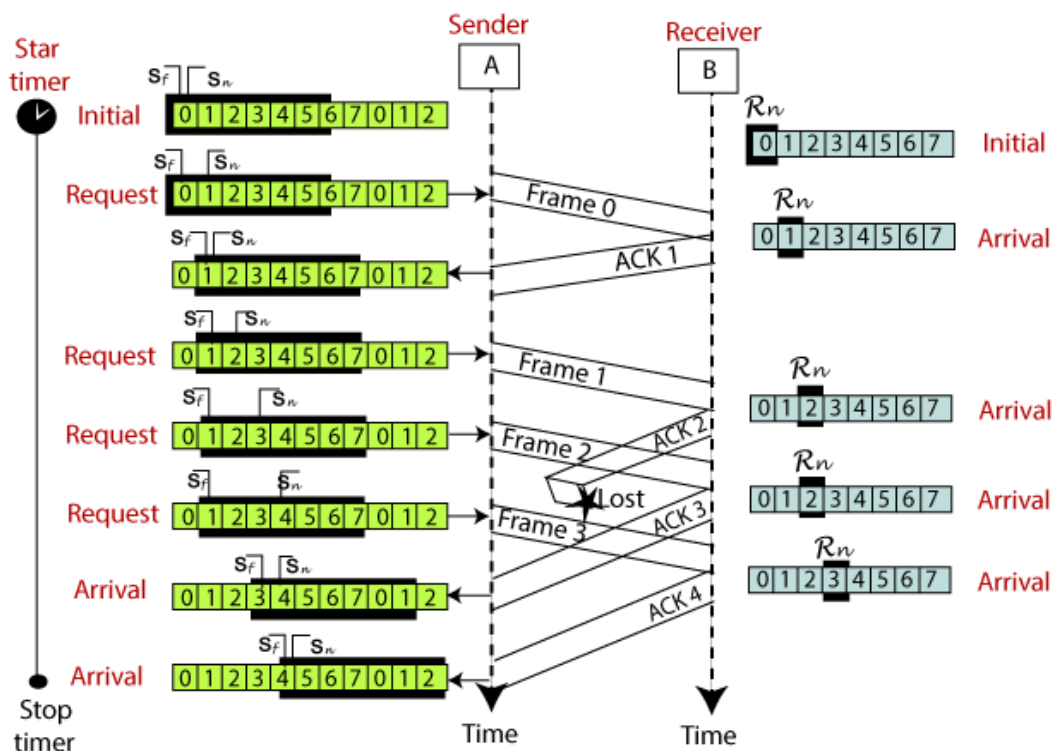
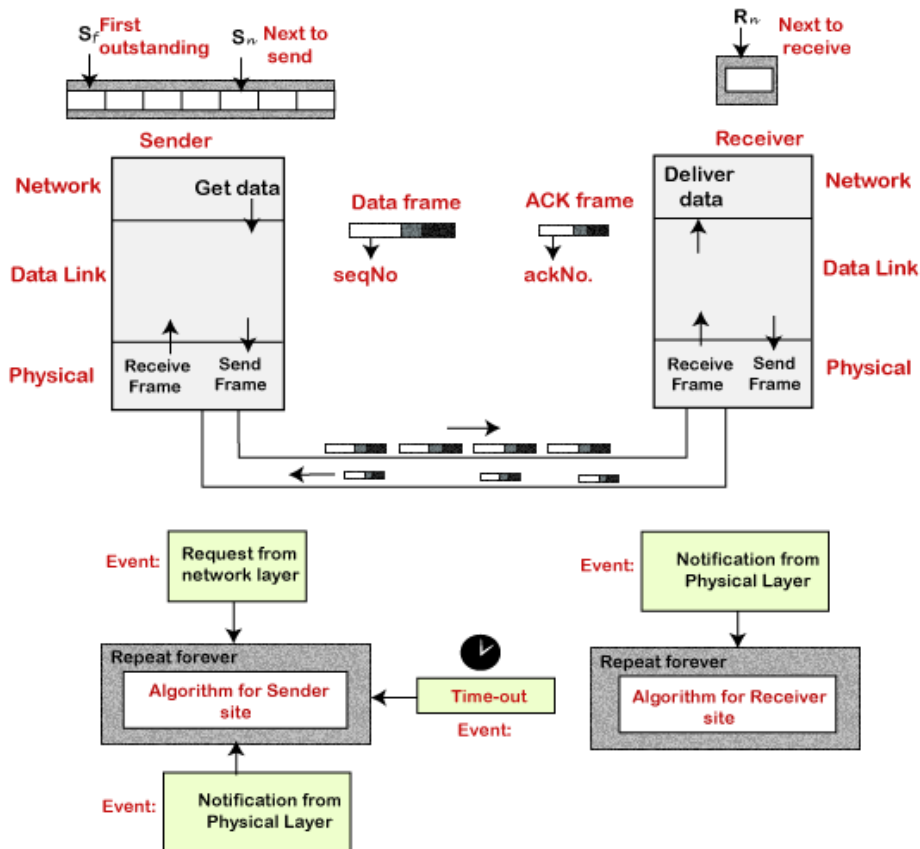
Working Principle

Go – Back – N ARQ provides for sending multiple frames before receiving the acknowledgment for the first frame. The frames are sequentially numbered and a finite number of frames. The maximum number of frames that can be sent depends upon the size of the sending window. If the acknowledgment of a frame is not received within an agreed upon time period, all frames starting from that frame are retransmitted.

The size of the sending window determines the sequence number of the outbound frames. If the sequence number of the frames is an n-bit field, then the range of sequence numbers that can be assigned is 0 to 2^n-1 . Consequently, the size of the sending window is 2^n-1 . Thus in order to accommodate a sending window size of 2^n-1 , a n-bit sequence number is chosen.

The sequence numbers are numbered as modulo-n. For example, if the sending window size is 4, then the sequence numbers will be 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, and so on. The number of bits in the sequence number is 2 to generate the binary sequence 00, 01, 10, 11.

The size of the receiving window is 1.



Selective Repeat Protocol

Selective repeat protocol, also called Selective Repeat ARQ (Automatic Repeat request), is a data link layer protocol that uses sliding window method for reliable

delivery of data frames. Here, only the erroneous or lost frames are retransmitted, while the good frames are received and buffered.

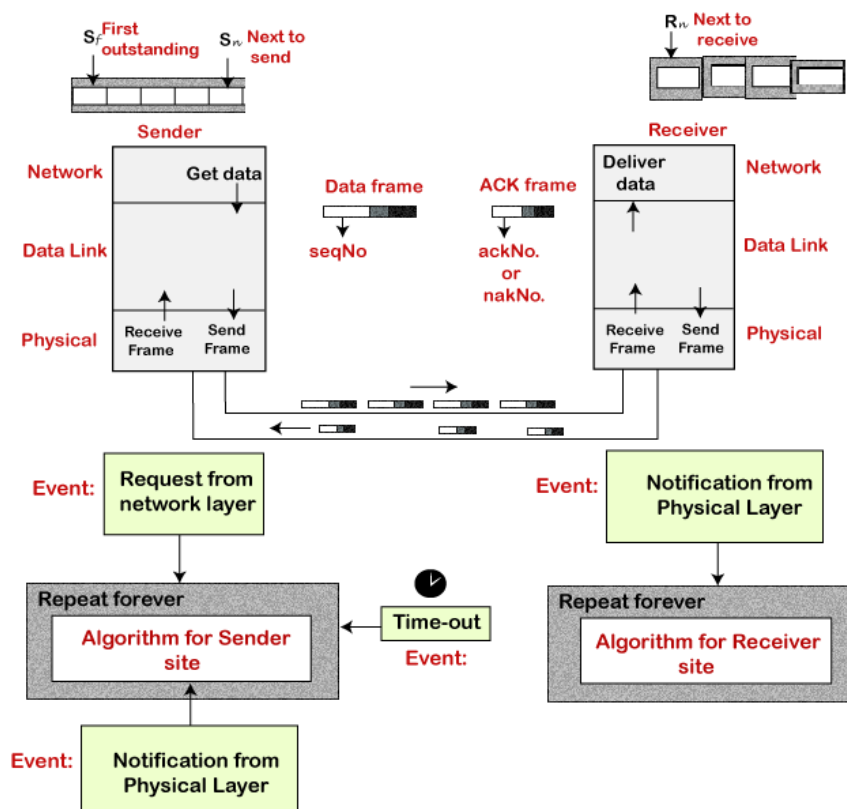
It uses two windows of equal size: a sending window that stores the frames to be sent and a receiving window that stores the frames received by the receiver. The size is half the maximum sequence number of the frame. For example, if the sequence number is from 0 – 15, the window size will be 8.

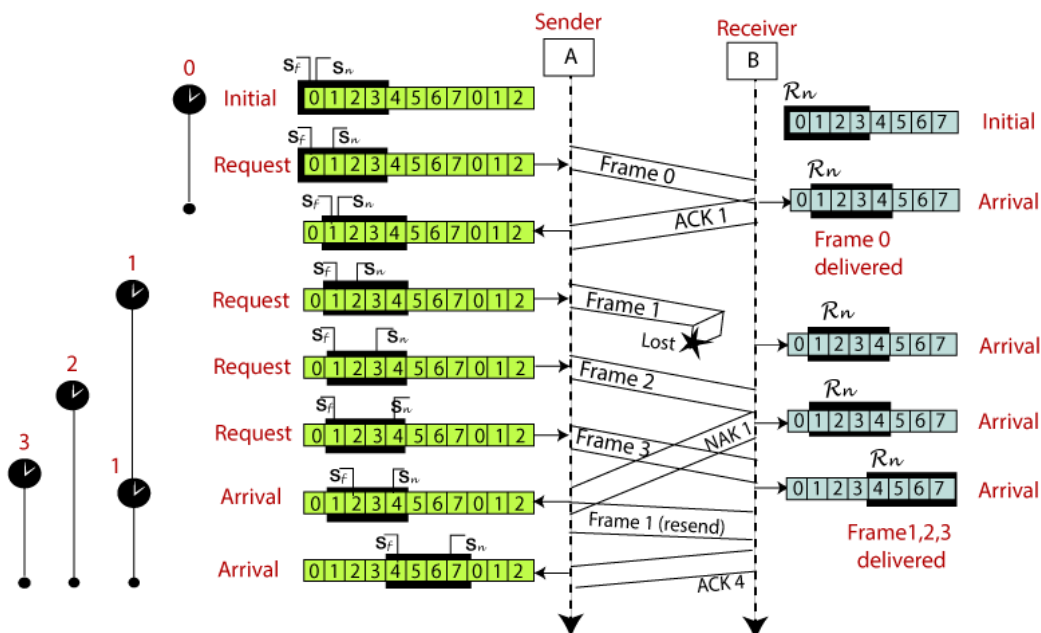
Working Principle

Selective Repeat protocol provides for sending multiple frames depending upon the availability of frames in the sending window, even if it does not receive acknowledgement for any frame in the interim. The maximum number of frames that can be sent depends upon the size of the sending window.

The receiver records the sequence number of the earliest incorrect or unreceived frame. It then fills the receiving window with the subsequent frames that it has received. It sends the sequence number of the missing frame along with every acknowledgement frame.

The sender continues to send frames that are in its sending window. Once, it has sent all the frames in the window, it retransmits the frame whose sequence number is given by the acknowledgements. It then continues sending the other frames.





Difference between Go Back-N and selective repeat protocol

Go-Back-N Protocol

In Go-Back-N Protocol, if the sent frame are find suspected then all the frames are re-transmitted from the lost packet to the last packet transmitted.

1.

Sender window size of Go-Back-N Protocol is N.

2.

Receiver window size of Go-Back-N Protocol is 1.

3.

Go-Back-N Protocol is less complex.

4.

In Go-Back-N Protocol, neither sender nor at receiver need sorting.

5.

In Go-Back-N Protocol, type of Acknowledgement is cumulative.

6.

Selective Repeat Protocol

In selective Repeat protocol, only those frames are re-transmitted which are found suspected.

Sender window size of selective Repeat protocol is also N.

Receiver window size of selective Repeat protocol is N.

Selective Repeat protocol is more complex.

In selective Repeat protocol, receiver side needs sorting to sort the frames.

In selective Repeat protocol, type of Acknowledgement is individual.

- | | |
|---|---|
| <p>7. In Go-Back-N Protocol, Out-of-Order packets are NOT Accepted (discarded) and the entire window is re-transmitted.</p> | <p>In selective Repeat protocol, Out-of-Order packets are Accepted.</p> |
| <p>8. In Go-Back-N Protocol, if Receiver receives a corrupt packet, then also, the entire window is re-transmitted.</p> | <p>In selective Repeat protocol, if Receiver receives a corrupt packet, it immediately sends a negative acknowledgement and hence only the selective packet is retransmitted.</p> |
| <p>9. Efficiency of Go-Back-N Protocol is $N/(1+2*a)$</p> | <p>Efficiency of selective Repeat protocol is also $N/(1+2*a)$</p> |

PROGRAM (C++)- Go Back-N protocol

```
#include <iostream>
using namespace std;
```

```
bool sender(int frame_number, int total_frames, int time_taken, int max_time, int size, int
*last_sent_frame_number, int *total_transmissions){
    if(time_taken <= max_time){
        cout<<"Acknowledgement Received for Frame number "<<frame_number<<endl;
        if(*last_sent_frame_number + 1 < total_frames){
            *last_sent_frame_number += 1;
            cout<<"Frame number "<<*last_sent_frame_number<<" --- Sent"<<endl;
            *total_transmissions += 1;
        }
        return true;
    }
    else{
        cout<<"No Acknowledgement Received"<<endl;
        return false;
    }
}
```

```
bool sender1(int frame_number, int total_frames, int time_taken, int max_time, int size, int
*last_sent_frame_number, int *total_transmissions){
    cout<<"Information sent from Sender's side\n";
    int temp_frame_number = frame_number;
    for(int i=0; i<size; i++){
        if(temp_frame_number < total_frames){
            cout<<"Frame number "<<temp_frame_number<<" --- Sent"<<endl;
            *last_sent_frame_number = temp_frame_number;
            temp_frame_number++;
        }
    }
}
```

```

        *total_transmissions += 1;
    }
    else{
        break;
    }
}cout<<endl;
    if(time_taken <= max_time){
        cout<<"Acknowledgement Received for Frame number "<<frame_number<<endl;
        if(*last_sent_frame_number + 1 < total_frames){
            *last_sent_frame_number += 1;
            cout<<"Frame number "<<*last_sent_frame_number<<" --- Sent"<<endl;
            *total_transmissions += 1;
        }
        return true;
    }
    else{
        cout<<"No Acknowledgement Received "<<endl;
        return false;
    }
}

int main() {
    cout<<"Enter total number of frames ";
    int n; cin>>n;    cout<<endl;
    cout<<"Enter window size ";
    int size; cin>>size;    cout<<endl;
    cout<<"Enter maximum time it will take to receive the Acknowledgement (in sec)";
    int max_time; cin>>max_time; cout<<endl<<endl;
    bool tag=false;
    int frame_number=0;
    int last_sent_frame_number = -1;
    int total_transmissions=0;
    while(frame_number < n){
        cout<<"Frame number "<<frame_number<<endl;
        cout<<"Enter time it will take to receive the Acknowledgement (in sec) ";
        int time_taken; cin>>time_taken;    cout<<endl<<endl;
        if(tag == false){
            tag = sender1(frame_number, n, time_taken, max_time, size,
&last_sent_frame_number, &total_transmissions);
        }
        else{
            tag = sender(frame_number, n, time_taken, max_time, size,
&last_sent_frame_number, &total_transmissions);
        }
        if(tag == true){

```

```

        frame_number++;
    }
    cout<<"=====\\n\\n";
}
cout<<endl<<"Total Transmissions= "<<total_transmissions;
return 0;
}

```

OUTPUT- Go Back-N protocol

Enter total number of frames 8

Enter window size 3

Enter maximum time it will take to receive the Acknowledgement (in sec)

Frame number 0

Enter time it will take to receive the Acknowledgement (in sec) 8

Information sent from Sender's side

Frame number 0 --- Sent

Frame number 1 --- Sent

Frame number 2 --- Sent

No Acknowledgement Received

=====

Frame number 0

Enter time it will take to receive the Acknowledgement (in sec) 2

Information sent from Sender's side

Frame number 0 --- Sent

Frame number 1 --- Sent

Frame number 2 --- Sent

Acknowledgement Received for Frame number 0

Frame number 3 --- Sent

=====

Frame number 1

Enter time it will take to receive the Acknowledgement (in sec) 1

Acknowledgement Received for Frame number 1

Frame number 4 --- Sent

=====

Frame number 2

Enter time it will take to receive the Acknowledgement (in sec) 7

No Acknowledgement Received

=====

```

Frame number 2
Enter time it will take to receive the Acknowledgement (in sec) 6

Information sent from Sender's side
Frame number 2 --- Sent
Frame number 3 --- Sent
Frame number 4 --- Sent

Acknowledgement Received for Frame number 2
Frame number 5 --- Sent
=====

Frame number 3
Enter time it will take to receive the Acknowledgement (in sec) 4

Acknowledgement Received for Frame number 3
Frame number 6 --- Sent
=====

Frame number 4
Enter time it will take to receive the Acknowledgement (in sec) 9

No Acknowledgement Received
=====

Frame number 4
Enter time it will take to receive the Acknowledgement (in sec) 1

Information sent from Sender's side
Frame number 4 --- Sent
Frame number 5 --- Sent
Frame number 6 --- Sent

Acknowledgement Received for Frame number 4
Frame number 7 --- Sent
=====

Frame number 5
Enter time it will take to receive the Acknowledgement (in sec) 5

Acknowledgement Received for Frame number 5
=====

Frame number 6
Enter time it will take to receive the Acknowledgement (in sec) 3

Acknowledgement Received for Frame number 6
=====

Frame number 7
Enter time it will take to receive the Acknowledgement (in sec) 10

No Acknowledgement Received
=====

Frame number 7
Enter time it will take to receive the Acknowledgement (in sec) 1

Information sent from Sender's side
Frame number 7 --- Sent

Acknowledgement Received for Frame number 7
=====

Total Transmissions= 18
Exit code: 0 (normal program termination)

```

PROGRAM (C++)- Selective Repeat protocol

```
#include <iostream>
```

```
using namespace std;
```

```
bool sender(int frame_number, int total_frames, int time_taken, int max_time, int size, int
*last_sent_frame_number, int *total_transmissions){
    if(time_taken <= max_time){
        cout<<"Acknowledgement Received for Frame number "<<frame_number<<endl;
        if(*last_sent_frame_number + 1 < total_frames){
            *last_sent_frame_number += 1;
            cout<<"Frame number "<<*last_sent_frame_number<<" --- Sent"<<endl;
            *total_transmissions += 1;
        }
        return true;
    }
    else{
        cout<<"No Acknowledgement Received"<<endl;
        return false;
    }
}
```

```
bool sender1(int frame_number, int total_frames, int time_taken, int max_time, int size, int
*last_sent_frame_number, int *total_transmissions){
    cout<<"Information sent from Sender's side\n";
    int temp_frame_number = frame_number;
    for(int i=0; i<size; i++){
        if(temp_frame_number < total_frames){
            cout<<"Frame number "<<temp_frame_number<<" --- Sent"<<endl;
            *last_sent_frame_number = temp_frame_number;
            temp_frame_number++;
            *total_transmissions += 1;
        }
        else{
            break;
        }
    }cout<<endl;
```

```
if(time_taken <= max_time){
    cout<<"Acknowledgement Received for Frame number "<<frame_number<<endl;
    if(*last_sent_frame_number + 1 < total_frames){
        *last_sent_frame_number += 1;
        cout<<"Frame number "<<*last_sent_frame_number<<" --- Sent"<<endl;
        *total_transmissions += 1;
```

```

    }
    return true;
}
else{
    cout<<"No Acknowledgement Received "<<endl;
    return false;
}
}

```

```

bool sender2(int frame_number, int total_frames, int time_taken, int max_time, int size, int
*last_sent_frame_number, int *total_transmissions){
    cout<<"Information sent from Sender's side\n";
    int temp_frame_number = frame_number;
    cout<<"Frame number "<<temp_frame_number<<" --- Sent"<<endl;
    *total_transmissions += 1;
    cout<<endl;
    if(time_taken <= max_time){
        cout<<"Acknowledgement Received for Frame number "<<frame_number<<endl;
        return true;
    }
    else{
        cout<<"No Acknowledgement Received"<<endl;
        return false;
    }
}

```

```

bool sender3(int frame_number, int total_frames, int time_taken, int max_time, int size, int
*last_sent_frame_number, int *total_transmissions){
    if(time_taken <= max_time){
        cout<<"Acknowledgement Received for Frame number "<<frame_number<<endl;
        return true;
    }
    else{
        cout<<"No Acknowledgement Received"<<endl;
        return false;
    }
}

```

```

int main() {
    cout<<"Enter total number of frames ";
    int n; cin>>n;    cout<<endl;
    cout<<"Enter window size ";

```



```

int size; cin>>size;  cout<<endl;
cout<<"Enter maximum time it will take to receive the Acknowledgement (in sec) ";
int max_time; cin>>max_time; cout<<endl<<endl;
bool tag=false;
int flag = 0;
int frame_number=0;
int last_sent_frame_number = -1;
int total_transmissions=0;

while(frame_number < n){
    cout<<"Frame number "<<frame_number<<endl;
    cout<<"Enter time it will take to receive the Acknowledgement (in sec) ";
    int time_taken;  cin>>time_taken;  cout<<endl<<endl;
    if(flag == 0){
        if(tag == false){
            tag = sender1(frame_number, n, time_taken, max_time, size,
&last_sent_frame_number, &total_transmissions);
        }
        else{
            tag = sender(frame_number, n, time_taken, max_time, size,
&last_sent_frame_number, &total_transmissions);
        }
    }
    else{
        if(tag == false){
            tag = sender2(frame_number, n, time_taken, max_time, size,
&last_sent_frame_number, &total_transmissions);
        }
        else{
            tag = sender3(frame_number, n, time_taken, max_time, size,
&last_sent_frame_number, &total_transmissions);
        }
    }
    if(tag == false || flag == 1){
        flag = 1;
    }
    if(tag == true){
        frame_number++;
        if(last_sent_frame_number < frame_number){
            flag = 0;
            tag = false;
        }
    }
}
cout<<"=====\n\n";
}

```

```

    cout<<endl<<"Total Transmissions= "<<total_transmissions;
    return 0;
}

```

OUTPUT- Selective Repeat protocol

```

Enter total number of frames 8

Enter window size 3

Enter maximum time it will take to receive the Acknowledgement (in sec) 6

Frame number 0
Enter time it will take to receive the Acknowledgement (in sec) 8

Information sent from Sender's side
Frame number 0 --- Sent
Frame number 1 --- Sent
Frame number 2 --- Sent

No Acknowledgement Received
=====

Frame number 0
Enter time it will take to receive the Acknowledgement (in sec) 2

Information sent from Sender's side
Frame number 0 --- Sent

Acknowledgement Received for Frame number 0
=====

Frame number 1
Enter time it will take to receive the Acknowledgement (in sec) 1

Acknowledgement Received for Frame number 1
=====

Frame number 2
Enter time it will take to receive the Acknowledgement (in sec) 7

No Acknowledgement Received
=====

```

Frame number 2
Enter time it will take to receive the Acknowledgement (in sec) 6

Information sent from Sender's side
Frame number 2 --- Sent

Acknowledgement Received for Frame number 2
=====

Frame number 3
Enter time it will take to receive the Acknowledgement (in sec) 4

Information sent from Sender's side
Frame number 3 --- Sent
Frame number 4 --- Sent
Frame number 5 --- Sent

Acknowledgement Received for Frame number 3
Frame number 6 --- Sent
=====

Frame number 4
Enter time it will take to receive the Acknowledgement (in sec) 9

No Acknowledgement Received
=====

Frame number 4
Enter time it will take to receive the Acknowledgement (in sec) 1

Information sent from Sender's side
Frame number 4 --- Sent

Acknowledgement Received for Frame number 4
=====

Frame number 5
Enter time it will take to receive the Acknowledgement (in sec) 5

Acknowledgement Received for Frame number 5

Frame number 6
Enter time it will take to receive the Acknowledgement (in sec) 3

Acknowledgement Received for Frame number 6
=====

Frame number 7
Enter time it will take to receive the Acknowledgement (in sec) 10

Information sent from Sender's side
Frame number 7 --- Sent

No Acknowledgement Received
=====

Frame number 7
Enter time it will take to receive the Acknowledgement (in sec) 1

Information sent from Sender's side
Frame number 7 --- Sent

Acknowledgement Received for Frame number 7
=====

Total Transmissions= 12
Exit code: 0 (normal program termination)

Experiment-8

Date-May 27, 2021.

AIM- Find shortest path between two nodes in a computer network using Dijkstra's shortest path algorithm.

Dijkstra Shortest path algorithm

This is a single-source shortest path algorithm and aims to find solution to the given problem statement. This algorithm works for both directed and undirected graphs. It works only for connected graphs. The graph should not contain negative edge weights. The algorithm predominantly follows Greedy approach for finding locally optimal solution. But, it also uses Dynamic Programming approach for building globally optimal solution, since the previous solutions are stored and further added to get final distances from the source vertex.

The main logic of this algorithm is based on the following formula- **$\text{dist}[r] = \min(\text{dist}[r], \text{dist}[q] + \text{cost}[q][r])$**

This formula states that distance vertex r , which is adjacent to vertex q , will be updated if and only if the value of $\text{dist}[q] + \text{cost}[q][r]$ is less than $\text{dist}[r]$. Here-

- Dist is a 1-D array which, at every step, keeps track of the shortest distance from source vertex to all other vertices.
- Cost is a 2-D array, representing the cost adjacency matrix for the graph.
- This formula uses both Greedy and Dynamic approaches. The Greedy approach is used for finding the minimum distance value, whereas the Dynamic approach is used for combining the previous solutions (**$\text{dist}[q]$** is already calculated and is used to calculate **$\text{dist}[r]$**).

Algorithm-

Step 1; Set $\text{dist}[s]=0$, $S=\varnothing$.

Step 2: For all nodes v except s , set $\text{dist}[v]=\infty$

Step 3: find q not in S such that $\text{dist}[q]$ is minimum.

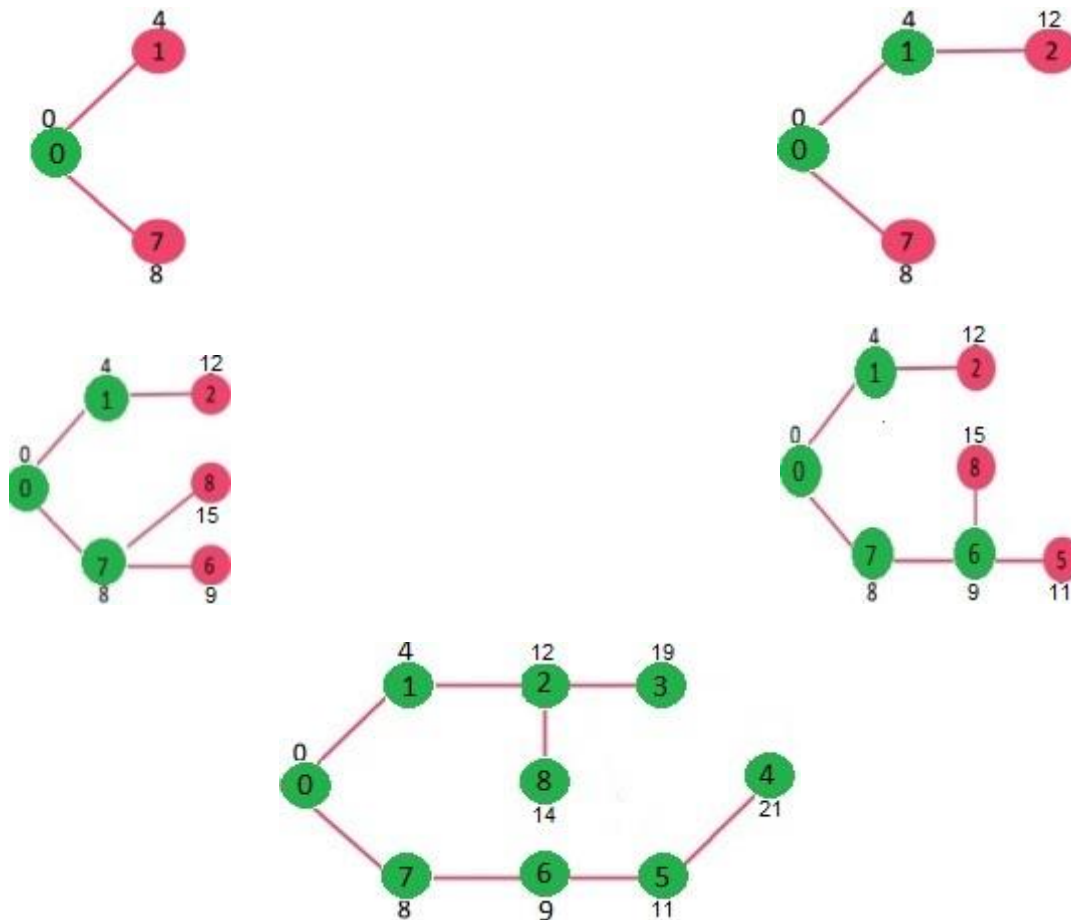
Step 4: add q to S .

Step 5: update $\text{dist}[r]$ for all r adjacent to q such that r is not in S //vertex r should not be visited $\text{dist}[r] = \min(\text{dist}[r], \text{dist}[q] + \text{cost}[q][r])$.

Step 6: Repeat Steps 3 to 5 until all the nodes are in S . Repeat till all the vertices have been visited.

Step 7: Print array dist having shortest path from the source vertex u to all other vertices.

Step 8: Exit.



PROGRAM (JAVA)-

```
import java.util.*;
import java.lang.*;
import java.io.*;
```

```
public class Dijkstra {
    static final int V = 9;
```

```
    static int minDistance(int dist[], Boolean sptSet[]) {
```

```

int min = Integer.MAX_VALUE, min_index = -1;

for (int v = 0; v < V; v++)
    if (sptSet[v] == false && dist[v] <= min) {
        min = dist[v];
        min_index = v;
    }

return min_index;
}

static void dijkstra(int graph[][], int src , int dest) {
    int dist[] = new int[V];
    Boolean sptSet[] = new Boolean[V];
    for (int i = 0; i < V; i++) {
        dist[i] = Integer.MAX_VALUE;
        sptSet[i] = false;
    }

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] != 0 && dist[u] != Integer.MAX_VALUE &&
dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    for (int i = 0; i < V; i++)
        if(i == dest)
            System.out.println("\nDistance from " + src + " to " + i + " : " + dist[i]);
    }

public static void main(String[] args) {

```

```

int graph[][] = new int[][] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                                { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                                { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                                { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                                { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                                { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                                { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                                { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                                { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

```

```

Scanner scn = new Scanner(System.in);
int src = 0, dest = 1;
System.out.print("Enter the source node : ");
src = scn.nextInt();
System.out.print("Enter the destination node : ");
dest = scn.nextInt();
dijkstra(graph, src, dest);
}
}

```

OUTPUT-

```

Enter the source node : 1
Enter the destination node : 6

Distance from 1 to 6 : 12

```

Experiment-9

Date-June 03, 2021.

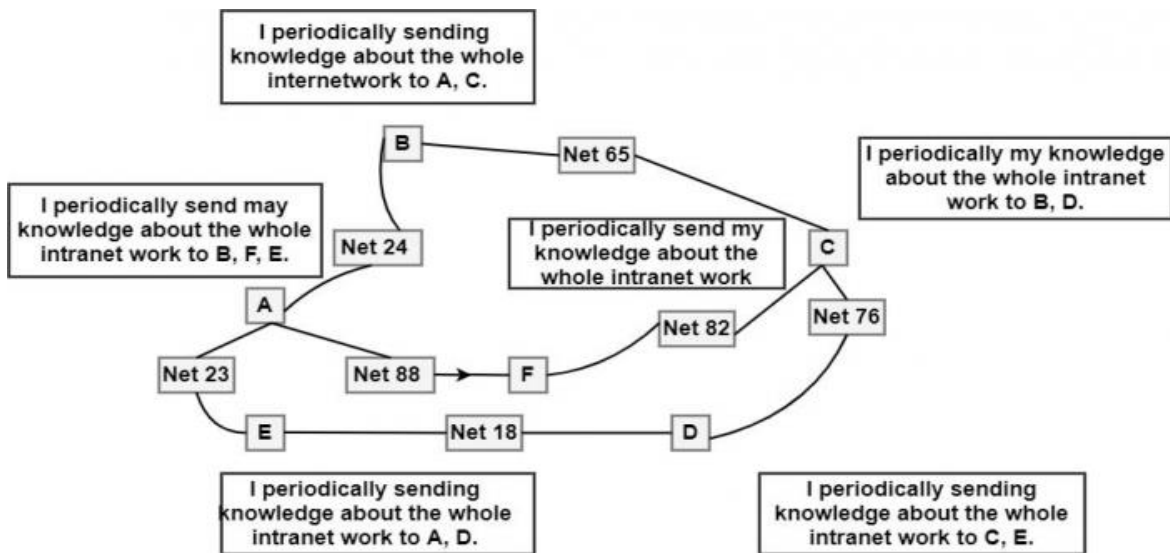
AIM- Implementation of distance vector routing algorithm.

Distance Vector Routing Algorithm

A distance-vector routing (DVR) protocol requires that a router inform its neighbors of topology changes periodically. Historically known as the old ARPANET routing algorithm. A distance-vector routing protocol in data networks determines the best route for data packets based on distance. Distance-vector routing protocols measure the distance by the number of routers a packet has to pass, one router counts as one hop. Some distance-vector protocols also take into account network latency and other factors that influence traffic on a given route. To determine the best route across a network, routers, on which a distance-vector protocol is implemented, exchange information with one another, usually routing tables plus hop counts for destination networks and possibly other traffic information. Distance-vector routing protocols also require that a router informs its neighbors of network topology changes periodically. Distance-vector routing protocols use the Bellman–Ford algorithm to calculate the best route. Another way of calculating the best route across a network is based on link cost, and is implemented through link-state routing protocols. The Distance vector algorithm is a dynamic algorithm. It is mainly used in ARPANET, and RIP. Each router maintains a distance table known as Vector.

The Distance vector algorithm is iterative, asynchronous and distributed.

- **Distributed:** It is distributed in that each node receives information from one or more of its directly attached neighbors, performs calculation and then distributes the result back to its neighbors.
- **Iterative:** It is iterative in that its process continues until no more information is available to be exchanged between neighbors.
- **Asynchronous:** It does not require that all of its nodes operate in the lock step with each other.



Advantages of Distance Vector routing –

- It is simpler to configure and maintain than link state routing.

Disadvantages of Distance Vector routing –

- It is slower to converge than link state.
- It is at risk from the count-to-infinity problem.

It creates more traffic than link state since a hop count change must be propagated to all routers and processed on each router.

PROGRAM (JAVA)-

```
import java.io.*;
import java.util.*;
public class A {
    static int graph[][];
    static int via[][];
    static int rt[][];
    static int v;
    static int e;

    public static void main(String args[]) {
        Scanner scn = new Scanner(System.in);
        v = 4;
        e = 5;
        graph = new int[v][v];
        via = new int[v][v];
        rt = new int[v][v];
    }
}
```

```

for (int i = 0; i < v; i++)
    for (int j = 0; j < v; j++)
        if (i == j)
            graph[i][j] = 0;
        else
            graph[i][j] = 9999;
for (int i = 0; i < e; i++) {
    System.out.println("\nPlease enter data for Edge " + (i + 1) + ":");
    System.out.print("Source: ");
    int s = scn.nextInt();
    System.out.print("Destination: ");
    int d = scn.nextInt();
    System.out.print("Cost: ");
    int c = scn.nextInt();
    graph[s - 1][d - 1] = c;
    graph[d - 1][s - 1] = c;
}

```

```

System.out.print("\nEnter Source Node : ");
int src = scn.nextInt();
System.out.print("Enter destination node : ");
int dest = scn.nextInt();
dvr_calc_disp("\nThe initial min distance b/w src and dest : ", src - 1, dest -
1);
System.out.print("\nPlease enter the Source Node for the edge whose cost
has changed: ");
int s = scn.nextInt();
System.out.print("Please enter the Destination Node for the edge whose
cost has changed: ");
int d = scn.nextInt();
System.out.print("Please enter the new cost: ");
int c = scn.nextInt();
graph[s - 1][d - 1] = c;
graph[d - 1][s - 1] = c;
dvr_calc_disp("\nThe final min distance b/w src and dest : ", src - 1, dest - 1);
}

```

```

static void dvr_calc_disp(String message, int src, int dest) {
    init_tables();
    update_tables();
    System.out.print(message);
    print_tables(src, dest);
}

static void update_table(int source) {
    for (int i = 0; i < v; i++)
        if (graph[source][i] != 9999) {
            int dist = graph[source][i];
            for (int j = 0; j < v; j++) {
                int inter_dist = rt[i][j];
                if (via[i][j] == source)
                    inter_dist = 9999;
                if (dist + inter_dist < rt[source][j]) {
                    rt[source][j] = dist + inter_dist;
                    via[source][j] = i;
                }
            }
        }
}

static void update_tables() {
    int k = 0;
    for (int i = 0; i < 4 * v; i++) {
        update_table(k);
        k++;
        if (k == v)    k = 0;
    }
}

static void init_tables() {
    for (int i = 0; i < v; i++)
        for (int j = 0; j < v; j++) {
            if (i == j) {
                rt[i][j] = 0;
                via[i][j] = i;
            } else {

```

```

        rt[i][j] = 9999;
        via[i][j] = 100;
    }
}

static void print_tables(int src, int dest) {
    for (int i = 0; i < v; i++)
        for (int j = 0; j < v; j++)
            if (i == src && j == dest)
                System.out.print(rt[i][j]);
}
}

```

OUTPUT-

```

Please enter data for Edge 1:
Source: 1
Destination: 2
Cost: 1

Please enter data for Edge 2:
Source: 2
Destination: 3
Cost: 3

Please enter data for Edge 3:
Source: 1
Destination: 3
Cost: 3

Please enter data for Edge 4:
Source: 3
Destination: 4
Cost: 2

Please enter data for Edge 5:
Source: 2
Destination: 4
Cost: 2

Enter Source Node : 1
Enter destination node : 4

The initial min distance b/w src and dest : 3
Please enter the Source Node for the edge whose cost has changed: 2
Please enter the Destination Node for the edge whose cost has changed: 4
Please enter the new cost: 7

The final min distance b/w src and dest : 5

```

Experiment-10

Date-June 10, 2021.

AIM- Implementation of Link State routing algorithm.

Link State Routing Algorithm

Link state routing is the second family of routing protocols. While distance vector routers use a distributed algorithm to compute their routing tables, link-state routing uses link-state routers to exchange messages that allow each router to learn the entire network topology. Based on this learned topology, each router is then able to compute its routing table by using a shortest path computation. It is a dynamic routing algorithm in which each router shares knowledge of its neighbours with every other router in the network. A router sends its information about its neighbours only to all the routers through flooding. Information sharing takes place only whenever there is a change. It makes use of **Dijkstra's Algorithm** for making routing tables. **Problems are** heavy traffic due to flooding of packets and flooding can result in infinite looping which can be solved by using **Time to live (TTL)** field.

Features of link state routing protocols:

- **Link state packet** – A small packet that contains routing information.
- **Link state database** – Collection information gathered from link state packet.
- **Shortest path first algorithm (Dijkstra algorithm)** – A calculation performed on the database results into shortest path
- **Routing table** – A list of known paths and interfaces.

Calculation of shortest path:

To find shortest path, each node need to run the famous **Dijkstra algorithm**. This famous algorithm uses the following steps:

1. The node is taken and chosen as a root node of the tree, this creates the tree with a single node, and now set the total cost of each node to some value based on the information in Link State Database.
2. Now the node selects one node, among all the nodes not in the tree like structure, which is nearest to the root, and adds this to the tree. The shape of the tree gets changed.
3. After this node is added to the tree, the cost of all the nodes not in the tree needs to be updated because the paths may have been changed.

4. The node repeats the Step 2 and Step 3 until all the nodes are added in the tree.

The three keys to understand the Link State Routing algorithm:

- **Knowledge about the neighborhood:** Instead of sending its routing table, a router sends the information about its neighborhood only. A router broadcast its identities and cost of the directly attached links to other routers.
- **Flooding:** Each router sends the information to every other router on the internetwork except its neighbors. This process is known as Flooding. Every router that receives the packet sends the copies to all its neighbors. Finally, each and every router receives a copy of the same information.
- **Information sharing:** A router sends the information to every other router only when the change occurs in the information.

Link State Routing has two phases:

1. Reliable Flooding

Initial state: Each node knows the cost of its neighbors.

Final state: Each node knows the entire graph.

2. Route Calculation

Each node uses Dijkstra's algorithm on the graph to calculate the optimal routes to all nodes. The Link state routing algorithm is also known as Dijkstra's algorithm which is used to find the shortest path from one node to every other node in the network.

The Dijkstra's algorithm is an iterative, and it has the property that after k^{th} iteration of the algorithm, the least cost paths are well known for k destination nodes.

Link State protocols in comparison to Distance Vector protocols have:

- It requires large amount of memory.
- Shortest path computations require many CPU cycles.
- If network use the little bandwidth ; it quickly reacts to topology changes
- All items in the database must be sent to neighbors to form link state packets.
- All neighbors must be trusted in the topology.
- Authentication mechanisms can be used to avoid undesired adjacency and problems.
- No split horizon techniques are possible in the link state routing.

BASIS FOR COMPARISON	DISTANCE VECTOR ROUTING	LINK STATE ROUTING
Algorithm	Bellman ford	Dijkstra
Network view	Topology information from the neighbour point of view	Complete information on the network topology
Best path calculation	Based on the least number of hops	Based on the cost
Updates	Full routing table	Link state updates
Updates frequency	Periodic updates	Triggered updates
CPU and memory	Low utilisation	Intensive
Simplicity	High simplicity	Requires a trained network administrator

BASIS FOR COMPARISON	DISTANCE VECTOR ROUTING	LINK STATE ROUTING
Convergence time	Moderate	Fast
Updates	On broadcast	On multicast
Hierarchical structure	No	Yes
Intermediate Nodes	No	Yes

PROGRAM (C++)-

```

#include <iostream>
#include <bits/stdc++.h>
using namespace std;
#define INF INT_MAX

void print_route(vector < int >const & prev, int i) {
    if (i < 0)
        return;
    print_route(prev, prev[i]);
    cout << i << " ";
}

int find_vertex(int v, int *distance, bool *visited){
    int min = INF, min_vertex = -1;
    for(int i=0; i<v; i++)

```



```

        if(visited[i] == false && distance[i] < min){
            min_vertex = i;
            min = distance[i];
        }
    return min_vertex;
}

```

```

void dijkistra(int v, int source, vector<vector<int>> vec, int destination){
    bool *visited = new bool[v];
    int *distance = new int[v];
    for(int i=0; i<v; i++){
        visited[i] = false;
        distance[i] = INF;
    }
    vector<int> last;
    for(int i=0; i<v; i++){
        last.push_back(-1);
    }
    distance[source] = 0;
    for(int i=0; i<v; i++){
        int vertex = find_vertex(v, distance, visited);
        visited[vertex] = true;
        for(int j=0; j<v; j++)
            if(visited[j] != true && vec[vertex][j] != INF){
                int num = distance[vertex] + vec[vertex][j];
                if(num < distance[j]){
                    distance[j] = num;
                    last[j] = vertex;
                }
            }
    }
}

```

```

if(destination == -1){
    for (int i = 0; i < v; i++) {
        cout << "Path (" << source << " —> " << i << "): Minimum cost = "<<
distance[i] << ", Route = [ ";
        print_route(last, i);
        cout << "]" << endl;
    }
}

```

```

        cout<<endl<<endl;
    }
    else{
        cout << "Path (" << source << " —> " << destination << "): Minimum cost =
"<< distance[destination] << ", Route = [ ";
        print_route(last, destination);
        cout << "]" << endl;
    }
}

```

```

int main() {
    cout<<"Enter vertices \n";
    int v; cin>>v;
    cout<<"Enter edges \n";
    int e; cin>>e;
    vector<vector<int>> vec( v , vector<int> (v));
    for(int i=0; i<v; i++)
        for(int j=0; j<v; j++)
            vec[i][j] = INF;

    cout<<"Enter edges' source, destination and weight \n";
    for(int i=0; i<e; i++){
        int x,y,d; cin>>x>>y>>d;
        vec[x][y] = d;
        vec[y][x] = d;
    }

    for(int i=0; i<v; i++)
        dijkistra(v, i, vec, -1);

    int source, destination;
    cout<<"Enter source and destination \n";
    cin>>source>>destination;
    dijkistra(v, source, vec, destination);

    return 0;
}

```

OUTPUT-

```
Enter vertices
4
Enter edges
6
Enter edges' source, destination and weight
1 2 3
0 1 4
0 2 5
2 3 1
3 0 7
1 3 6
Path (0 → 0): Minimum cost = 0, Route = [ 0 ]
Path (0 → 1): Minimum cost = 4, Route = [ 0 1 ]
Path (0 → 2): Minimum cost = 5, Route = [ 0 2 ]
Path (0 → 3): Minimum cost = 6, Route = [ 0 2 3 ]

Path (1 → 0): Minimum cost = 4, Route = [ 1 0 ]
Path (1 → 1): Minimum cost = 0, Route = [ 1 ]
Path (1 → 2): Minimum cost = 3, Route = [ 1 2 ]
Path (1 → 3): Minimum cost = 4, Route = [ 1 2 3 ]

Path (2 → 0): Minimum cost = 5, Route = [ 2 0 ]
Path (2 → 1): Minimum cost = 3, Route = [ 2 1 ]
Path (2 → 2): Minimum cost = 0, Route = [ 2 ]
Path (2 → 3): Minimum cost = 1, Route = [ 2 3 ]

Path (3 → 0): Minimum cost = 6, Route = [ 3 2 0 ]
Path (3 → 1): Minimum cost = 4, Route = [ 3 2 1 ]
Path (3 → 2): Minimum cost = 1, Route = [ 3 2 ]
Path (3 → 3): Minimum cost = 0, Route = [ 3 ]

Enter source and destination
1 3
Path (1 → 3): Minimum cost = 4, Route = [ 1 2 3 ]

...Program finished with exit code 0
Press ENTER to exit console.□
```

Experiment-11

Date-June 17, 2021.

AIM- Write a program for congestion control using Leaky bucket algorithm.

In the network layer, before the network can make Quality of service guarantees, it must know what traffic is being guaranteed. One of the main causes of congestion is that traffic is often bursty.

To understand this concept first we have to know little about traffic shaping. Traffic Shaping is a mechanism to control the amount and the rate of the traffic sent to the network. Approach of congestion management is called Traffic shaping. Traffic shaping helps to regulate rate of data transmission and reduces congestion. There are 2 types of traffic shaping algorithms:

- Leaky Bucket

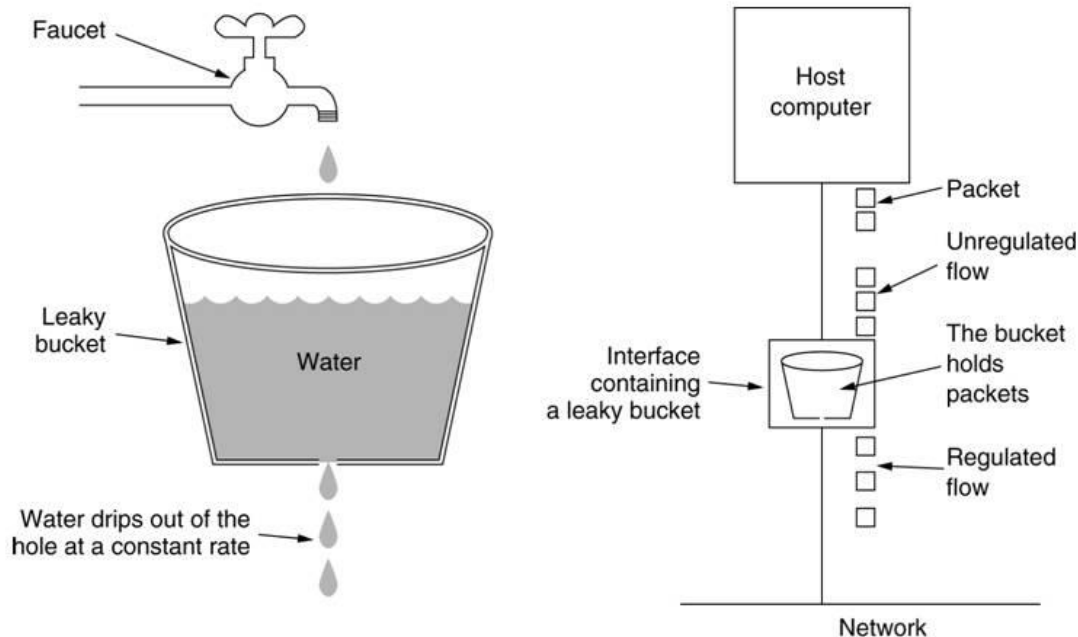
- Token Bucket

Suppose we have a bucket in which we are pouring water in a random order but we have to get water in a fixed rate, for this we will make a hole at the bottom of the bucket. It will ensure that water coming out is in a some fixed rate, and also if bucket will full we will stop pouring in it.

The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate.

In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. The use of the leaky bucket shapes the input traffic to make it conform to this commitment. In Figure the host sends a burst of data at a rate of 12 Mbps for 2 s, for a total of 24 Mbits of data. The host is silent for 5 s and then sends data at a rate of 2 Mbps for 3 s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in 10 s. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10 s.

Leaky Bucket



(a) A leaky bucket with water. (b) a leaky bucket with packets.

Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the leaky bucket may prevent congestion.

A simple leaky bucket algorithm can be implemented using FIFO queue. A FIFO queue holds the packets. If the traffic consists of fixed-size packets (e.g., cells in ATM networks), the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.

The following is an algorithm for variable-length packets:

Initialize a counter to n at the tick of the clock.

If n is greater than the size of the packet, send the packet and decrement the counter by the packet size. Repeat this step until n is smaller than the packet size.

Reset the counter and go to step 1.

Example – Let $n=1000$

Packet =

200	700	500	450	400	200
-----	-----	-----	-----	-----	-----

Since $n > \text{front of Queue}$ i.e. $n > 200$

Therefore, $n = 1000 - 200 = 800$

Packet size of 200 is sent to the network.

200	700	500	450	400
-----	-----	-----	-----	-----

Now Again $n > \text{front of the queue}$ i.e. $n > 400$

Therefore, $n = 800 - 400 = 400$

Packet size of 400 is sent to the network.

200	700	500	450
-----	-----	-----	-----

Since $n < \text{front of queue}$

Therefore, the procedure is stop.

Initialize $n = 1000$ on another tick of clock.

This procedure is repeated until all the packets are sent to the network.

PROGRAM (JAVA)-

```
import java.util.*;
```

```
import java.io.*;
```

```
public class Main {
```

```
    public static void main(String args[]) {
```

```
        int n, sto, bs, ips, ops, sl;
```

```
        Scanner sc = new Scanner(System.in);
```

```
        // Initial total number of packets that are passing into the bucket
```

```
        System.out.print("Enter total number of queries entering : ");
```

```
        n = sc.nextInt();
```

```
        // first estimate total packets in the bucket
```

```
        System.out.print("\nEnter the total number of packets in the bucket : ");
```

```
        sto = sc.nextInt();
```

```
        // the total number of packets that can be filled in the bucket
```

```

    System.out.print("\nEnter total number of packets that can be filled in the
bucket : ");
    bs = sc.nextInt();
    // number of input packets at a particular time
    System.out.print("\nEnter total number of packets that are inputed into the
bucket : ");
    ips = sc.nextInt();
    // no. of packets that quit the bucket
    System.out.print("\nEnter total number of packets that are comming out
the bucket : ");
    ops = sc.nextInt();
    System.out.println();

    for (int i = 0; i < n; i++) {
        System.out.println("At time: " + (i + 1));
        // total size left
        sl = bs - sto;
        System.out.println("size left " + sl);
        if (ips <= (sl)) {
            sto += ips;
            System.out.println("Packets received= " + sto + " out of bucket size= " +
bs);
            System.out.println("Packet loss is 0 ");
        } else {
            System.out.println("Packet loss = " + (ips - (sl)));
            // if the bucket is at filled fully
            sto = bs;
            System.out.println("Buffer size= " + sto + " out of bucket size= " + bs);
        }
        System.out.println("output sent " + ops);
        sto -= ops;
        System.out.println("After output sent storage is " + sto);
        System.out.println("-----");
    }
}
}
}

```

OUTPUT-

```
Enter total number of queries entering : 4

Enter the total number of packets in the bucket : 15

Enter total number of packets that can be filled in the bucket : 7

Enter total number of packets that are inputed into the bucket : 4

Enter total number of packets that are coming out the bucket : 3

At time: 1
size left -8
Packet loss = 12
Buffer size= 7 out of bucket size= 7
output sent 3
After output sent storage is 4
-----
At time: 2
size left 3
Packet loss = 1
Buffer size= 7 out of bucket size= 7
output sent 3
After output sent storage is 4
-----
At time: 3
size left 3
Packet loss = 1
Buffer size= 7 out of bucket size= 7
output sent 3
After output sent storage is 4
-----
At time: 4
size left 3
Packet loss = 1
Buffer size= 7 out of bucket size= 7
output sent 3
After output sent storage is 4
-----
```


Experiment-12

Date-June 24, 2021.

AIM- Implementation of Data encryption and decryption (Program for Caesar Cipher and Generalized Caesar Cipher).

THEORY- Caesar Cipher is a mono-alphabetic cipher wherein each letter of the plaintext is substituted by another letter to form the ciphertext. It is a simplest form of substitution cipher scheme. This cryptosystem is generally referred to as the Shift Cipher. The concept is to replace each alphabet by another alphabet which is 'shifted' by some fixed number between 0 and 25.

For this type of scheme, both sender and receiver agree on a 'secret shift number' for shifting the alphabet. This number which is between 0 and 25 becomes the key of encryption. The name 'Caesar Cipher' is occasionally used to describe the Shift Cipher when the 'shift of three' is used.

Process

- In order to encrypt a plaintext letter, the sender positions the sliding ruler underneath the first set of plaintext letters and slides it to LEFT by the number of positions of the secret shift.
- The plaintext letter is then encrypted to the ciphertext letter on the sliding ruler underneath. The result of this process is depicted in the following illustration for an agreed shift of three positions. In this case, the plaintext 'tutorial' is encrypted to the ciphertext 'wxwrueldo'. Here is the ciphertext alphabet for a Shift of 3 –

Plaintext Alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Alphabet	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- On receiving the ciphertext, the receiver who also knows the secret shift, positions his sliding ruler underneath the ciphertext alphabet and slides it to RIGHT by the agreed shift number, 3 in this case.
- He then replaces the ciphertext letter by the plaintext letter on the sliding ruler underneath. Hence the ciphertext 'wxwrueldo' is decrypted to 'tutorial'. To decrypt a message encoded with a Shift of 3, generate the plaintext alphabet using a shift of '-3' as shown below –

Ciphertext Alphabet	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Plaintext Alphabet	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w

- For decryption just follow the reverse of encryption process.

PROGRAM (JAVA)-

```
import java.util.Scanner;

public class CaesarCipher {
    public static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";

    public static String encrypt(String plainText, int shiftKey) {
        plainText = plainText.toLowerCase();
        String cipherText = "";
        for (int i = 0; i < plainText.length(); i++) {
            int charPosition = ALPHABET.indexOf(plainText.charAt(i));
            int keyVal = (shiftKey + charPosition) % 26;
            char replaceVal = ALPHABET.charAt(keyVal);
            cipherText += replaceVal;
        }
        return cipherText;
    }

    public static String decrypt(String cipherText, int shiftKey) {
        cipherText = cipherText.toLowerCase();
        String plainText = "";
        for (int i = 0; i < cipherText.length(); i++) {
            int charPosition = ALPHABET.indexOf(cipherText.charAt(i));
            int keyVal = (charPosition - shiftKey) % 26;
            if (keyVal < 0)
                keyVal = ALPHABET.length() + keyVal;
            char replaceVal = ALPHABET.charAt(keyVal);
            plainText += replaceVal;
        }
        return plainText;
    }
}
```

```

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the String for Encryption : ");
    String message = sc.next();
    System.out.print("Enter the key : ");
    int key = sc.nextInt();
    String encrypted = encrypt(message, key);
    String decrypted = decrypt(encrypted, key);
    System.out.println("\nEncrypted msg : " + encrypted);
    System.out.println("Decrypted msg : " + decrypted);
}
}

```

OUTPUT-

```

Enter the String for Encryption : Sarthak
Enter the key : 2

Encrypted msg : uctvjcm
Decrypted msg : sarthak
PS D:\temp> d:; cd 'd:\temp'; & 'c:\Users\sarthak\.vscode\
OpenJDK\jdk-11.0.11.9-hotspot\bin\java.exe' '-Dfile.encoding=
ae6fb942660e98a0007\redhat.java\jdt_ws\temp_72ed65cd\bin
Enter the String for Encryption : programming
Enter the key : 3

Encrypted msg : surjudpplqj
Decrypted msg : programming
PS D:\temp>

```