**PRACTICAL NO. 1**

**AIM:** Study of Network Topologies and LAN

**THEORY:**

Network Topology is the schematic description of a network arrangement, connecting various nodes (sender and receiver) through lines of connection.

**Types of Network topologies**

**BUS Topology**

Bus topology is a network type in which every computer and network device is connected to single cable. When it has exactly two endpoints, then it is called **Linear Bus topology**

**Features of Bus Topology**

* It transmits data only in one direction.
* Every device is connected to a single cable

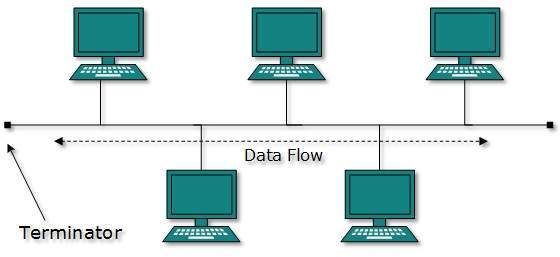


Fig (1): Bus Topology

**Advantages of Bus Topology**

* It is cost effective.
* Cable required is least compared to other network topology.
* Used in small networks.
* It is easy to understand.
* Easy to expand joining two cables together.

**Disadvantages of Bus Topology**

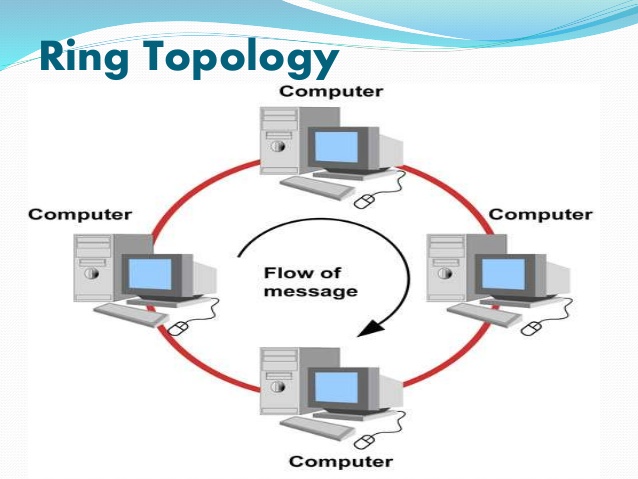
* Cables fails then whole network fails.
* If network traffic is heavy or nodes are more the performance of the network decreases.
* Cable has a limited length.
* It is slower than the ring topology.

**RING Topology**

It is called ring topology because it forms a ring as each computer is connected to another computer, with the last one connected to the first. Exactly two neighbours for each device.

**Features of Ring Topology**

* A number of repeaters are used for Ring topology with 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**.
* In Dual Ring Topology, two ring networks are formed, and data flow is in opposite direction in them. Also, if one ring fails, the second ring can act as a backup, to keep the network up.
* Data is transferred in a sequential manner that is bit by bit. Data transmitted, has to pass through each node of the network, till the destination node.



**Advantages of Ring Topology**

* Transmitting network is not affected by high traffic or by adding more nodes, as only the nodes having tokens can transmit data.
* Cheap to install and expand

**Disadvantages of Ring Topology**

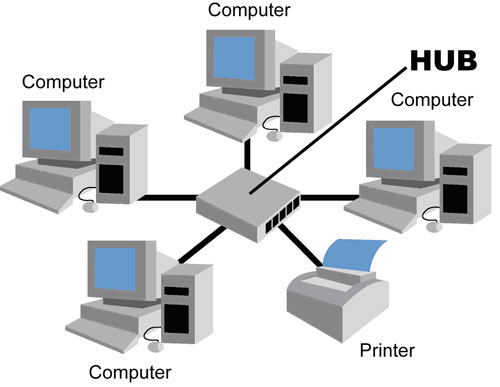
* Troubleshooting is difficult in ring topology.
* Adding or deleting the computers disturbs the network activity.
* Failure of one computer disturbs the whole network.

**STAR Topology**

In this type of topology all the computers are connected to a single hub through a cable. This hub is the central node and all others nodes are connected to the central node.

**Features of Star Topology**

* Every node has its own dedicated connection to the hub.
* Hub acts as a repeater for data flow.
* Can be used with twisted pair, Optical Fibre or coaxial cable.



**Advantages of Star Topology**

* Fast performance with few nodes and low network traffic.
* Hub can be upgraded easily.
* Easy to troubleshoot.
* Easy to setup and modify.
* Only that node is affected which has failed, rest of the nodes can work smoothly.

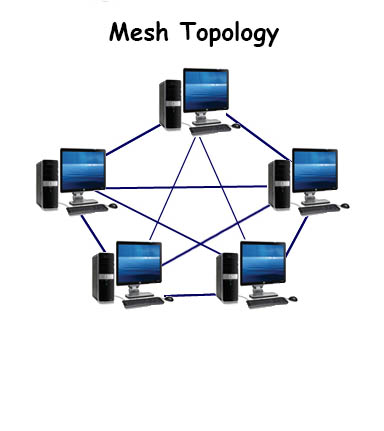
**Disadvantages of Star Topology**

* Cost of installation is high.
* Expensive to use.
* If the hub fails then the whole network is stopped because all the nodes depend on the hub.
* Performance is based on the hub that is it depends on its capacity

**MESH Topology**

It is a point-to-point connection to other nodes or devices. All the network nodes are connected to each other. Mesh has n(n-2)/2 physical channels to link n devices.

Routing and Flooding are two techniques to transmit data over the Mesh topology



**Routing**

In routing, the nodes have a routing logic, as per the network requirements. Like routing logic to direct the data to reach the destination using the shortest distance. Or, routing logic which has information about the broken links, and it avoids those node etc. We can even have routing logic, to re-configure the failed nodes.

**Flooding**

In flooding, the same data is transmitted to all the network nodes, hence no routing logic is required. The network is robust, and the its very unlikely to lose the data. But it leads to unwanted load over the network

**Features of Mesh Topology**

* Fully connected.
* Robust.
* Not flexible.

**Types of Mesh Topology**

* **Partial Mesh Topology:** In this topology some of the systems are connected in the same fashion as mesh topology but some devices are only connected to two or three devices.
* **Full Mesh Topology:** Each and every nodes or devices are connected to each other.

**Advantages of Mesh Topology**

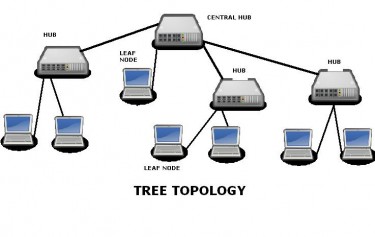
* Each connection can carry its own data load.
* It is robust.
* Fault is diagnosed easily.
* Provides security and privacy.

**Disadvantages of Mesh Topology**

* Installation and configuration is difficult.
* Cabling cost is more.
* Bulk wiring is required.

**TREE Topology**

It has a root node and all other nodes are connected to it forming a hierarchy. It is also called hierarchical topology. It should at least have three levels to the hierarchy.



**Features of Tree Topology**

* Ideal if workstations are located in groups.
* Used in Wide Area Network.

**Advantages of Tree Topology**

* Extension of bus and star topologies.
* Expansion of nodes is possible and easy.
* Easily managed and maintained.
* Error detection is easily done.

**Disadvantages of Tree Topology**

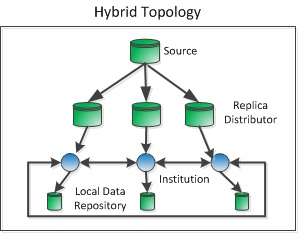
* Heavily cabled.
* Costly.
* If more nodes are added maintenance is difficult.
* Central hub fails, network fails.

**HYBRID Topology**

It is two different types of topologies which is a mixture of two or more topologies. For example if in an office in one department ring topology is used and in another star topology is used, connecting these topologies will result in Hybrid Topology (ring topology and star topology).

**Features of Hybrid Topology**

* It is a combination of two or topologies
* Inherits the advantages and disadvantages of the topologies included



**Advantages of Hybrid Topology**

* Reliable as Error detecting and trouble shooting is easy.
* Effective.
* Scalable as size can be increased easily.
* Flexible.

**Disadvantages of Hybrid Topology**

* Complex in design.
* Costly.

**CONCLUSION:**

Thus the various LAN topologies are studied.

**PRACTICAL NO. 2**

**AIM:** To study hardware and software devices in networking

**Theory:**



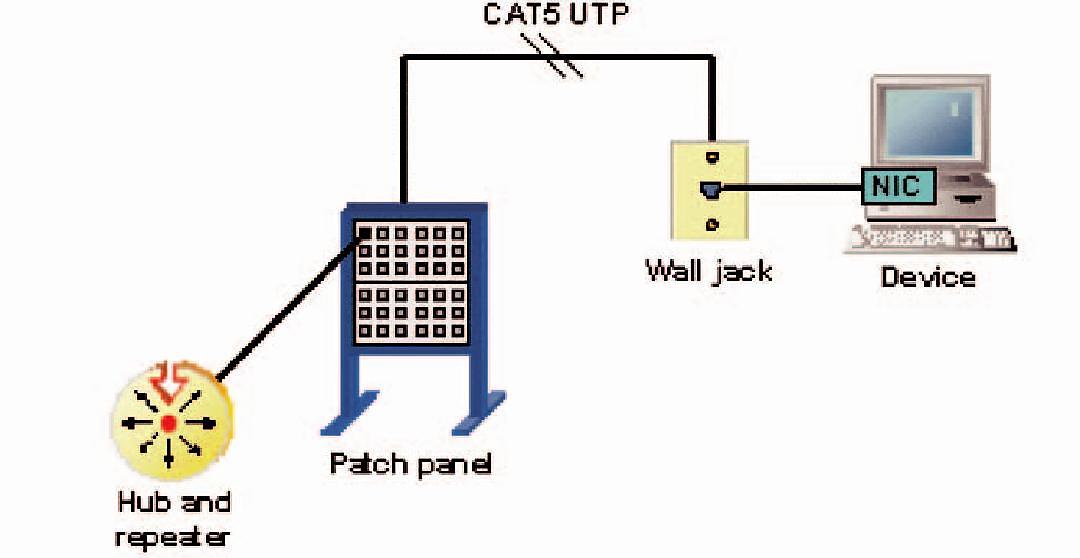
A networking model offers a generic means to separate computer networking functions into multiple layers. Each of these layers relies on the layers below it to provide supporting capabilities and performs support to the layers above it. Such a model of layered functionality is also called a “protocol stack” or “protocol suite”.

Protocols, or rules, can do their work in either hardware or software or, as with most protocol stacks, in a combination of the two. The nature of these stacks is that the lower layers do their work in hardware or firmware (software that runs on specific hardware chips) while the higher layers work in software.

The main benefits of the OSI model include the following:

* Helps users understand the big picture of networking
* Helps users understand how hardware and software elements function together
* Makes troubleshooting easier by separating networks into manageable pieces
* Defines terms that networking professionals can use to compare basic functional relationships on different networks
* Helps users understand new technologies as they are developed
* Aids in interpreting vendor explanations of product functionality

**Layer 1 – The Physical Layer**



The physical layer of the OSI model defines connector and interface specifications, as well as the medium (cable) requirements. Electrical, mechanical, functional, and procedural specifications are provided for sending a bit stream on a computer network.

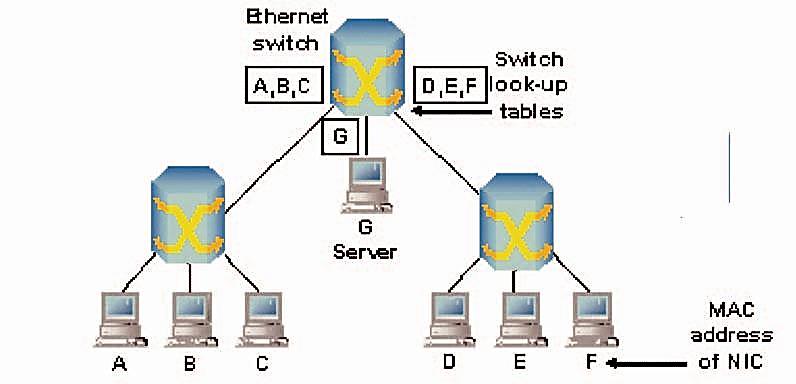
Components of the physical layer include:

* Cabling system components
* Adapters that connect media to physical interfaces
* Connector design and pin assignments
* Hub, repeater, and patch panel specifications
* Wireless system components
* Parallel SCSI (Small Computer System Interface)
* Network Interface Card (NIC)

In a LAN environment, Category 5e UTP (Unshielded Twisted Pair) cable is generally used for the physical layer for individual device connections. Fiber optic cabling is often used for the physical layer in a vertical or riser backbone link. The IEEE, EIA/TIA, ANSI, and other similar standards bodies developed standards for this layer.

**Note:** The Physical Layer of the OSI model is only part of a LAN (Local Area Network).

**Layer 2 – The Data Link Layer**



Layer 2 of the OSI model provides the following functions:

* Allows a device to access the network to send and receive messages
* Offers a physical address so a device’s data can be sent on the network
* Works with a device’s networking software when sending and receiving messages
* Provides error-detection capability

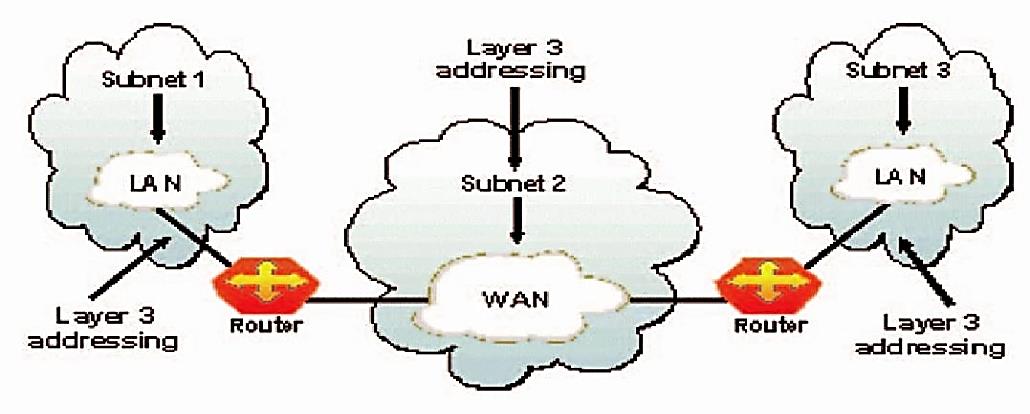
Common networking components that function at layer 2 include:

* Network interface cards
* Ethernet and Token Ring switches
* Bridges

NICs have a layer 2 or MAC address. A switch uses this address to filter and forward traffic, helping relieve congestion and collisions on a network segment.

Bridges and switches function in a similar fashion; however, bridging is normally a software program on a CPU, while switches use Application-Specific Integrated Circuits (ASICs) to perform the task in dedicated hardware, which is much faster.

**Layer 3 – The Network Layer**



Layer 3, the network layer of the OSI model, provides an end-to-end logical addressing system so that a packet of data can be routed across several layer 2 networks (Ethernet, Token Ring, Frame Relay, etc

Initially, software manufacturers, such as Novell, developed proprietary layer 3 addressing. However, the net-working industry has evolved to the point that it requires a common layer 3 addressing system. The Internet Protocol (IP) addresses make networks easier to both set up and connect with one another. The Internet uses IP addressing to provide connectivity to millions of networks around the world.

To make it easier to manage the network and control the flow of packets, many organizations separate their network layer addressing into smaller parts known as subnets. Routers use the network or subnet portion of the IP addressing to route traffic between different networks. Each router must be configured specifically for the networks or subnets that will be connected to its interfaces.

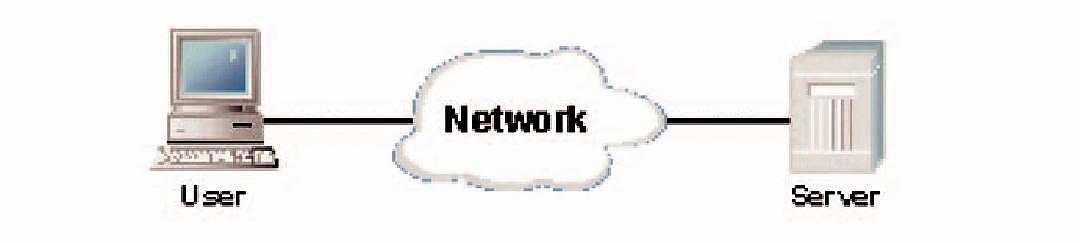
Routers communicate with one another using routing protocols, such as Routing Information Protocol (RIP) and Open version of Shortest Path First (OSPF), to learn of other networks that are present and to calculate the best way to reach each network based on a variety of criteria (such as the path with the fewest routers). Routers and other networked systems make these routing decisions at the network layer.

Two of the additional functions of the network layer are diagnostics and the reporting of logical variations in normal network operation. While the network layer diagnostics may be initiated by any networked system, the system discovering the variation reports it to the original sender of the packet that is found to be outside nor-mal network operation.

The variation reporting exception is content validation calculations. If the calculation done by the receiving sys-tem does not match the value sent by the originating system, the receiver discards the related packet with no report to the sender. Retransmission is left to a higher layer’s protocol.

Some basic security functionality can also be set up by filtering traffic using layer 3 addressing on routers or other similar devices.

**Layer 4 – The Transport Layer**

****

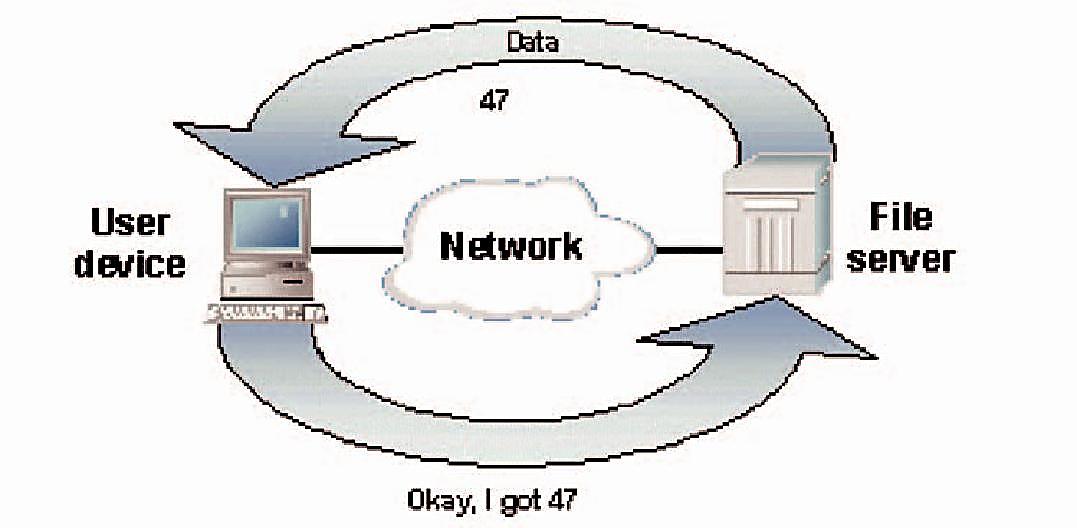
Layer 4, the transport layer of the OSI model, offers end-to-end communication between end devices through a network. Depending on the application, the transport layer either offers reliable, connection-oriented or connectionless, best-effort communications.

Some of the functions offered by the transport layer include:

* Application identification
* Client-side entity identification
* Confirmation that the entire message arrived intact
* Segmentation of data for network transport
* Control of data flow to prevent memory overruns
* Establishment and maintenance of both ends of virtual circuits
* Transmission-error detection
* Realignment of segmented data in the correct order on the receiving side
* Multiplexing or sharing of multiple sessions over a single physical link

The most common transport layer protocols are the connection-oriented TCP Transmission Control Protocol (TCP) and the connectionless UDP User Datagram Protocol (UDP).

**Layer 5 – The Session Layer**

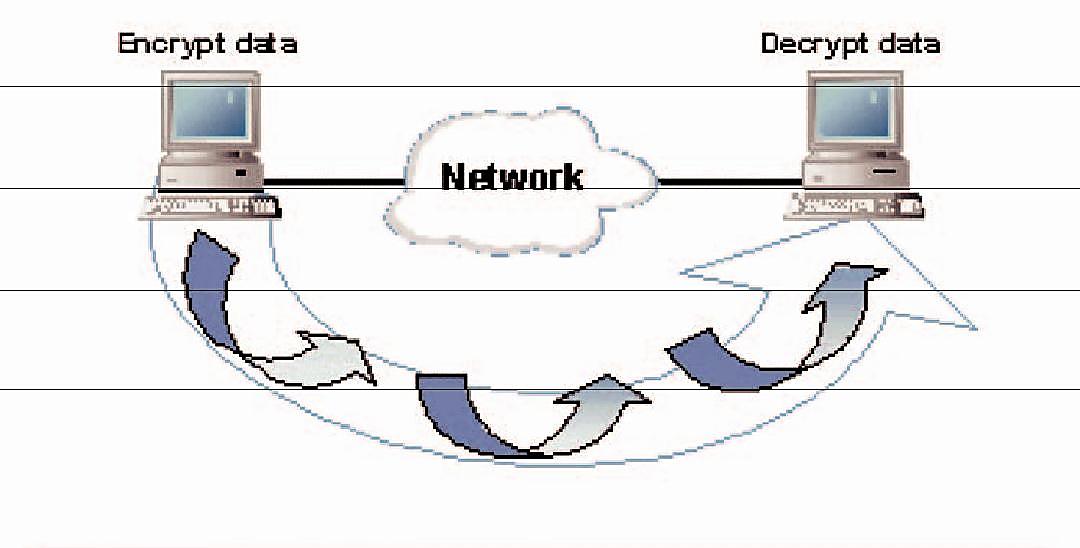


Layer 5, the session layer, provides various services, including tracking the number of bytes that each end of the session has acknowledged receiving from the other end of the session. This session layer allows applications functioning on devices to establish, manage, and terminate a dialog through a network.

Session layer functionality includes:

* Virtual connection between application entities
* Synchronization of data flow
* Creation of dialog units
* Connection parameter negotiations
* Partitioning of services into functional groups
* Acknowledgements of data received during a session
* Retransmission of data if it is not received by a device

**Layer 6 – The Presentation Layer**



Layer 6, the presentation layer, is responsible for how an application formats the data to be sent out onto the network. The presentation layer basically allows an application to read (or understand) the message.

Examples of presentation layer functionality include:

* Encryption and decryption of a message for security
* Compression and expansion of a message so that it travels efficiently
* Graphics formatting
* Content translation
* System-specific translation

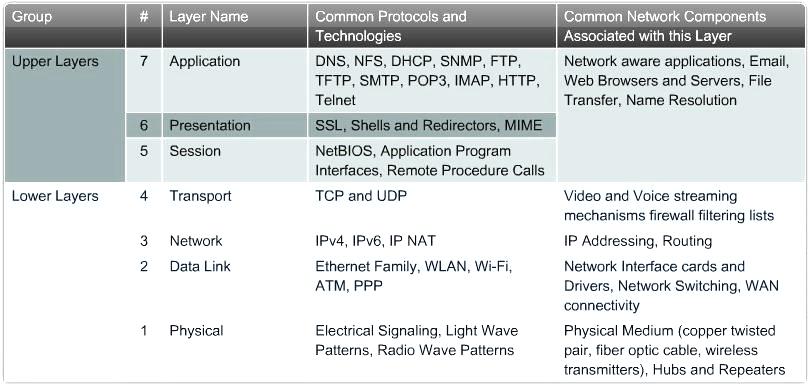
**Layer 7 – The Application Layer**



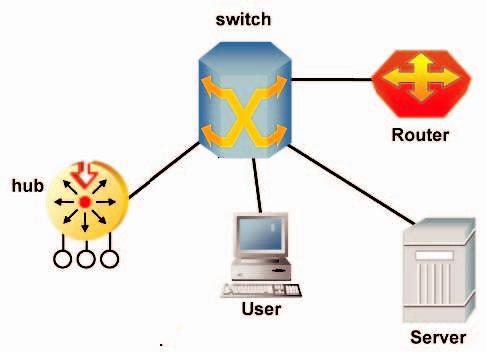
Layer 7, the application layer, provides an interface for the end user operating a device connected to a net-work. This layer is what the user sees, in terms of loading an application (such as Web browser or e-mail); that is, this application layer is the data the user views while using these applications.

Examples of application layer functionality include:

* Support for file transfers
* Ability to print on a network
* Electronic mail
* Electronic messaging
* Browsing the World Wide Web



Equipment at the Layers



Some of the layers use equipment to support the identified functions. Hub related activity is “Layer One”. The naming of some devices designates the functional layer such as “Layer Two Switch” or “Layer Three Switch”. Router functions focus on “Layer Three”. User workstations and servers are often identified with “Layer Seven”.

**CONCLUSION:**

Hence the different hardware and software devices used in networking are studied

**PRACTICAL NO.3**

**AIM:** Implementation of VRC

Theory:

Vertical redundancy check (VRC) is an error-checking method used on an eight-bit ASCII character. In VRC, a parity bit is attached to each byte of data, which is then tested to determine whether the transmission is correct. VRC is considered an unreliable error-detection method because it only works if an even number of bits is distorted.

A vertical redundancy check is also called a transverse redundancy check when used in combination with other error-controlling codes such as a longitudinal redundancy check.

VRC is a redundancy check meant for parallel synchronized bits applied one bit at a time. It uses additional parallel channels for check bits and refers to single-parity bits or larger hamming codes.

Although parities are only meant for error detection and not error correction, they still can remain part of a system for correcting errors.

**PROGRAM:(Can be implemented in C,C++ or Java)**

#include<stdio.h>

#include<conio.h>

int binary(int);

void parity(int[]);

int arr[9],arr1[9];

char chr;

int temp,temp1,i,j;

void main() {

char chr1;

clrscr();

printf("Enter Data :");

scanf("%c %c",&chr,&chr1);

temp=chr;

binary(temp);

printf("\nAscii value is : %d\n",temp);

printf("\nBinary Form : ");

for(i=0;i<8;i++) {

printf("%d ",arr1[i]);

}

printf("\n");

parity(arr1);

temp1=chr1;

binary(temp1);

printf("\n\nAscii value is : %d\n",temp1);

printf("\nBinary Form : ");

for(i=0;i<8;i++) {

printf("%d ",arr1[i]);

}

printf("\n"); parity(arr1);

getch();

}

void parity(int a[]) {

int count;

count=0;

for(i=0;i<8;i++) {

if(a[i]==1)

count++;

}

if(count%2==0)

a[8]=0;

else

a[8]=1;

count=0;

printf("Receiver Side :\n");

printf("\n\nVRC : \n");

for(i=0;i<9;i++) {

if(i==8)

printf(" | ");

printf("%d ",a[i]);

}

}

intbinary(int x) {

intrem,t;

inti=0;

while(x!=0) {

rem=x%2;

arr[i]=rem;

x=x/2;

i++;

}

t=8-i;

for(j=0;j<t;j++) {

arr1[j]=0;

}

for(j=t;j<=7;j++) {

i--;

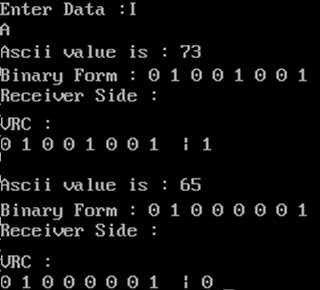
arr1[j]=arr[i];

}

return(0);

}

**OUTPUT:**



**Conclusion**: Thus we have implemented the VRC

**PRACTICAL NO. 4**

**AIM:** Implementation of Error Correction (CRC Code) using C/C++/JAVA.

**Theory:**

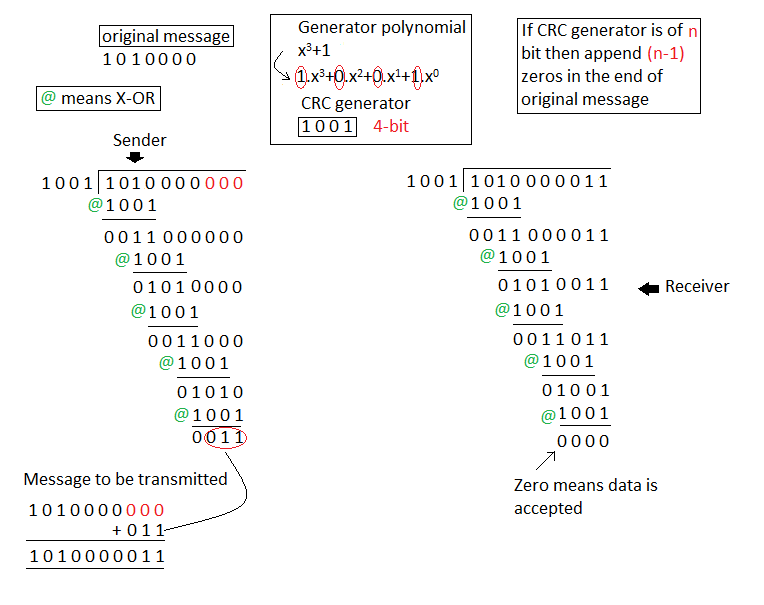
**Cyclic redundancy check (CRC)**

* Unlike checksum scheme, which is based on addition, CRC is based on binary division.
* In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
* At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.

A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

[](http://www.geeksforgeeks.org/wp-content/uploads/gq/2016/01/3.png)

**Example :**

[](http://www.geeksforgeeks.org/wp-content/uploads/gq/2016/01/4.png)

**PROGRAM:(Can be implemented in C,C++ or Java)**

import java.util.\*;

import java.util.\*;

class CRC {

public static void main(String args[]) {

Scanner scan = new Scanner(System.in);

int n;

//Accept the input

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

n = scan.nextInt();

int data[] = new int[n];

System.out.println("Enter the data, bit by bit:");

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

System.out.println("Enter bit number " + (n-i) + ":");

data[i] = scan.nextInt();

}

// Accept the divisor

System.out.println("Enter the size of the divisor:");

n = scan.nextInt();

int divisor[] = new int[n];

System.out.println("Enter the divisor, bit by bit:");

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

System.out.println("Enter bit number " + (n-i) + ":");

divisor[i] = scan.nextInt();

}

// Divide the inputted data by the inputted divisor

// Store the remainder that is returned by the method

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

for(int i=0 ; i < remainder.length-1 ; i++) {

System.out.print(remainder[i]);

}

System.out.println("\nThe CRC code generated is:");

for(int i=0 ; i < data.length ; i++) {

System.out.print(data[i]);

}

for(int i=0 ; i < remainder.length-1 ; i++) {

System.out.print(remainder[i]);

}

System.out.println();

// Create a new array

// It will have the remainder generated by the above method appended

// to the inputted data

int sent\_data[] = new int[data.length + remainder.length - 1];

System.out.println("Enter the data to be sent:");

for(int i=0 ; i < sent\_data.length ; i++) {

System.out.println("Enter bit number " + (sent\_data.length-i)+ ":");

sent\_data[i] = scan.nextInt();

}

receive(sent\_data, divisor);

}

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 array stores the remainder

remainder = new int[divisor.length];

// Initially, remainder's bits will be set to the data bits

System.arraycopy(data, 0, remainder, 0, divisor.length);

// Loop runs for same number of times as number of bits of data

// This loop will continuously exor the bits of the remainder and

// divisor

for(i=0 ; i < old\_data.length ; i++) {

System.out.println((i+1) + ".) First data bit is : "

+ remainder[0]);

System.out.print("Remainder : ");

if(remainder[0] == 1) {

// We have to exor the remainder bits with divisor bits

for(int j=1 ; j < divisor.length ; j++) {

remainder[j-1] = exor(remainder[j], divisor[j]);

System.out.print(remainder[j-1]);

}

}

else {

// We have to exor the remainder bits with 0

for(int j=1 ; j < divisor.length ; j++) {

remainder[j-1] = exor(remainder[j], 0);

System.out.print(remainder[j-1]);

}

}

// The last bit of the remainder will be taken from the data

// This is the 'carry' taken from the dividend after every step // of division

remainder[divisor.length-1] = data[i+divisor.length];

System.out.println(remainder[divisor.length-1]);

}

return remainder;

}

static int exor(int a, int b) {

// This simple function returns the exor of two bits

if(a == b) {

return 0;

}

return 1;

}

static void receive(int data[], int divisor[]) {

// This is the receiver method

// It accepts the data and divisor (although the receiver already has

// the divisor value stored, with no need for the sender to resend it)

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

// Division is done

for(int i=0 ; i < remainder.length ; i++) {

if(remainder[i] != 0) {

// If remainder is not zero then there is an error

System.out.println("There is an error in received data...");

return;

}

}

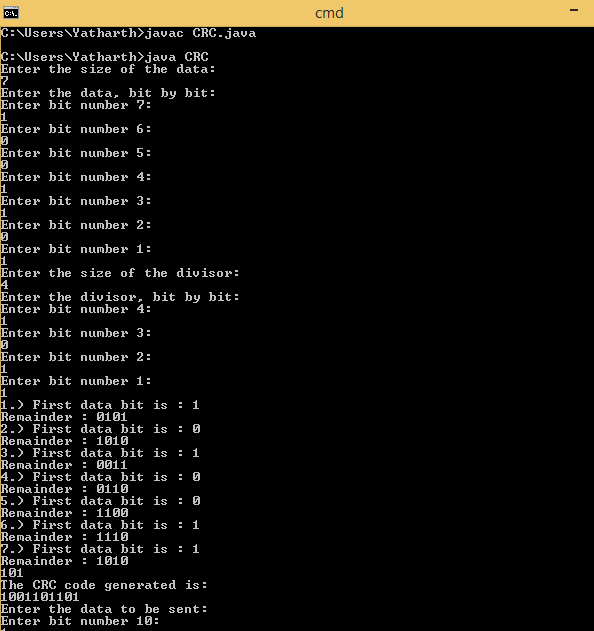
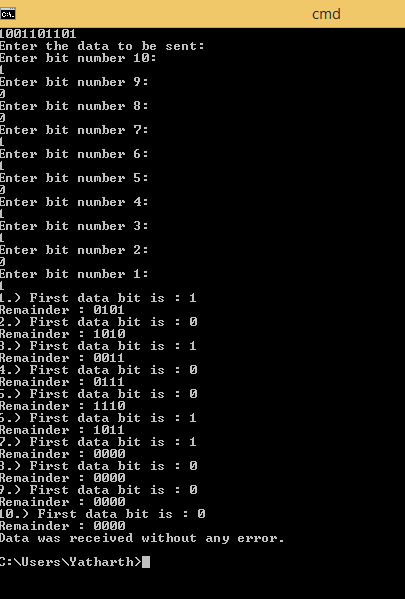
//Otherwise there is no error in the received data

System.out.println("Data was received without any error.");

}

}

**OUTPUT:**

**Conclusion**: Thus we have implemented the CRC

**PRACTICAL NO. 5**

**AIM:** Implementation of hamming code

**THEORY:**

The key to the Hamming Code is the use of extra parity bits to allow the identification of a single error. Create the code word as follows:

1. Mark all bit positions that are powers of two as parity bits. (positions 1, 2, 4, 8, 16, 32, 64, etc.)
2. All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.)
3. Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.   
   Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1,3,5,7,9,11,13,15,...)  
   Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (2,3,6,7,10,11,14,15,...)  
   Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (4,5,6,7,12,13,14,15,20,21,22,23,...)  
   Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15,24-31,40-47,...)  
   Position 16: check 16 bits, skip 16 bits, check 16 bits, skip 16 bits, etc. (16-31,48-63,80-95,...)  
   Position 32: check 32 bits, skip 32 bits, check 32 bits, skip 32 bits, etc. (32-63,96-127,160-191,...)  
   etc.
4. Set a parity bit to 1 if the total number of ones in the positions it checks is odd. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

Here is an example:

A byte of data: 10011010  
Create the data word, leaving spaces for the parity bits: \_ \_ 1 \_ 0 0 1 \_ 1 0 1 0  
Calculate the parity for each parity bit (a ? represents the bit position being set):

* Position 1 checks bits 1,3,5,7,9,11:  **?** \_ **1** \_ **0** 0 **1** \_ **1** 0 **1** 0. Even parity so set position 1 to a 0: **0** \_ **1** \_ **0** 0 **1** \_ **1** 0 **1** 0
* Position 2 checks bits 2,3,6,7,10,11:  
  0 **? 1** \_ 0 **0 1** \_ 1 **0 1** 0. Odd parity so set position 2 to a 1: 0 **1 1** \_ 0 **0 1** \_ 1 **0 1** 0
* Position 4 checks bits 4,5,6,7,12:  
  0 1 1 **? 0 0 1** \_ 1 0 1 **0**. Odd parity so set position 4 to a 1: 0 1 1 **1 0 0 1** \_ 1 0 1 **0**
* Position 8 checks bits 8,9,10,11,12:  
  0 1 1 1 0 0 1 **? 1 0 1 0**. Even parity so set position 8 to a 0: 0 1 1 1 0 0 1 **0 1 0 1 0**
* Code word: 011100101010.

**Program: (Can be implemented in c, c++,java)**

#include<iostream>

using namespace std;

int main() {

int data[10];

int dataatrec[10],c,c1,c2,c3,i;

cout<<"Enter 4 bits of data one by one\n";

cin>>data[0];

cin>>data[1];

cin>>data[2];

cin>>data[4];

//Calculation of even parity

data[6]=data[0]^data[2]^data[4];

data[5]=data[0]^data[1]^data[4];

data[3]=data[0]^data[1]^data[2];

cout<<"\nEncoded data is\n";

for(i=0;i<7;i++)

cout<<data[i];

cout<<"\n\nEnter received data bits one by one\n";

for(i=0;i<7;i++)

cin>>dataatrec[i];

c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];

c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];

c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];

c=c3\*4+c2\*2+c1 ;

if(c==0) {

cout<<"\nNo error while transmission of data\n";

}

else {

cout<<"\nError on position "<<c;

cout<<"\nData sent : ";

for(i=0;i<7;i++)

cout<<data[i];

cout<<"\nData received : ";

for(i=0;i<7;i++)

cout<<dataatrec[i];

cout<<"\nCorrect message is\n";

//if errorneous bit is 0 we complement it else vice versa

if(dataatrec[7-c]==0)

dataatrec[7-c]=1;

else

dataatrec[7-c]=0;

for (i=0;i<7;i++) {

cout<<dataatrec[i];

}

}

return 0;

}

**Output**

*Enter 4 bits of data one by one*  
*1*  
*0*  
*1*  
*0*

*Encoded data is*  
*1010010*

*Enter received data bits one by one*  
*1*  
*0*  
*1*  
*0*  
*0*  
*1*  
*0*

**Conclusion**: Thus we have implemented the hamming code

**PRACTICAL NO. 6**

**AIM:** Study of Packet Tracer installation & Simulation commands.

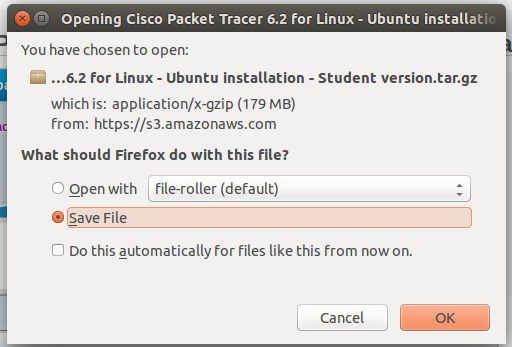
**THEORY:**

First step is to download Packet Tracer from Cisco Network Academy. To get access you must be a NetAcad student. The link will be listed under Resources. Look for the student version which is a fairly large file size.

When you find the version you’ll be sent to a new page. Right underneath the title of the download is the actual download button. Save this to your desktop.

[](http://www.packet6.com/wp-content/uploads/2015/07/how-to-install-packet-tracer-ubuntu-02.png)

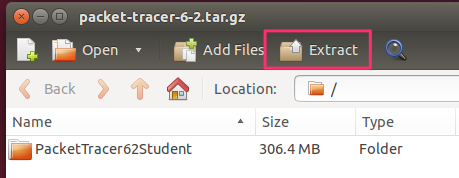
Packet Tracer is downloaded to your desktop in a compressed package.

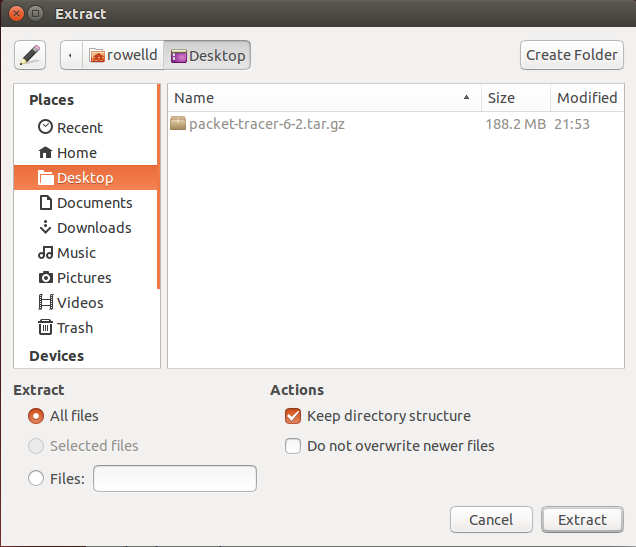
[](http://www.packet6.com/wp-content/uploads/2015/07/how-to-install-packet-tracer-ubuntu-03.png)

[](http://www.packet6.com/wp-content/uploads/2015/07/how-to-install-packet-tracer-ubuntu-04.png)

Open the package to reveal the contents. Using the GUI, double click on the Cisco Packet Tracer file to open it.

Within the package is a directory called **PacketTracer62Student**. We need to extract this directory and its contents to the Desktop. To extract the directory, click on the *Extract* button which is highlighted below.

[](http://www.packet6.com/wp-content/uploads/2015/07/how-to-install-packet-tracer-ubuntu-05.png)

[](http://www.packet6.com/wp-content/uploads/2015/07/how-to-install-packet-tracer-ubuntu-06.png)

Now lets open Terminal and use command line to complete the rest of the installation. First we change directories to where the Packet Tracer directory is stored, on your Desktop.

rowelld@rowelld-VirtualBox:~$ cd Desktop

rowelld@rowelld-VirtualBox:~/Desktop$ cd PacketTracer62Student/

Within this directory is an installation script, *Install*. This script, once initiated, will perform the installation of Packet Tracer. To begin installation type ./install in Terminal. Be sure to respond to each prompt.

rowelld@rowelld-VirtualBox:~/Desktop/PacketTracer62Student$ ./install

Welcome to Cisco Packet Tracer 6.2 Installation

Read the following End User License Agreement "EULA" carefully. You must accept the terms of this EULA to install and use Cisco Packet Tracer.

Press the Enter key to read the EULA.

Do you accept the terms of the EULA? (Y)es/(N)o

Y

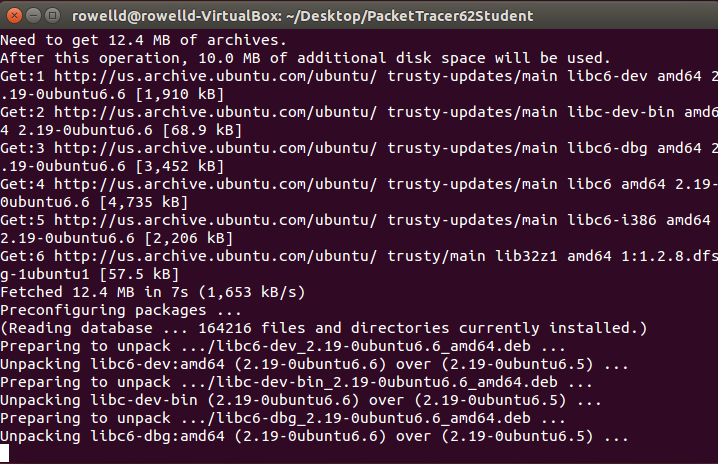
You have accepted the terms to the EULA. Congratulations. Packet Tracer will now be installed.

Enter location to install Cisco Packet Tracer or press enter for default [/opt/pt]: [enter]

Not able to create and copy files to /opt/pt

Should we try to gain root access with sudo? [Yn] Y

[sudo] password for rowelld:

[](http://www.packet6.com/wp-content/uploads/2015/07/how-to-install-packet-tracer-ubuntu-07.png)

Output from the command line during installation.

Should we create a symbolic link "packettracer" in /usr/local/bin for easy Cisco Packet Tracer startup? [Yn] Y

Type "packettracer" in a terminal to start Cisco Packet Tracer

Writing PT6HOME environment variable to /etc/profile

Cisco Packet Tracer 6.2 installed successfully

Packet Tracer is now fully installed and a symbolic link was created which is a shortcut to starting up Packet Tracer .To start Packet Tracer, type packet tracer into the command prompt.

rowelld@rowelld-VirtualBox:/$ packettracer

Starting Packet Tracer 6.2

On initial start, Packet Tracer will save your user files to **/home/*%username%*/pt** which can be changed by going to **Options > Preferences**.

[](http://www.packet6.com/wp-content/uploads/2015/07/how-to-install-packet-tracer-ubuntu-08.png)

After clicking OK, the Packet Tracer application opens and you will now be able to drag and drop routers and switches and begin practicing for the CCNA by completing labs.

1. **Router Configuration:-**

**Access CLI Prompt Of Router**:-

Router>enable

Router#configure terminal

Enter configuration commandss, one per. End with CNTL/Z.

Router(config) #

|  |  |  |  |
| --- | --- | --- | --- |
| **MODE** | **PROMPT** | **COMMAND TO ENTER** | **COMMAND TO EXIT** |
| User EXEC | Router > | Default mode after booting. Login with password, if configured. | Use **exit** command |
| Privileged EXEC | Router # | Use **enable** command from user exec mode | Use **exit** command |
| Global Configuration | Router(config)# | Use **configure terminal** command from privileged exec mode | Use **exit** command |
| Interface Configuration | Router(config-if)# | Use **interface type *number*** command from global configuration mode | Use **exit** command to return in global configuration mode |
| Sub-Interface Configuration | Router(config-subif) | Use **interface type *sub interface number*** command from global configuration mode or interface configure mode | Use **exit** to return previous mode. Use **end** command to return in privileged exec mode. |
| Setup | Parameter[Parameter value]: | Router will automatically insert in this mode if running configuration is not present | Press **CTRL+C** to abort. Type **yes** to save configuration, or **no** to exit without saving. |
| ROMMON | ROMMON > | Enter **reload** command from privileged exec mode. Press **CTRL + C** key combination during the first 60 seconds of booting process | Use **exit** command. |

1. IOS commands are not case sensitive; you can enter them in uppercase, lowercase, or mixed case. Password is case sensitive. Make sure you type it in correct case.
2. In any mode, you can obtain a list of commands available on that mode by entering a question mark (?).
3. Standard order of accessing mode is:User Exec mode => Privileged Exec mode => Global Configuration mode => Interface Configuration mode => Sub Interface Configuration mode
4. Router will enter in setup mode only if it fails to load a valid running configuration.
5. Router will enter in ROMMON mode only if it fails to load a valid IOS image file.

**Change Default Router Name:**

**Syntax:-**

Router(config) #hostname LAB1

LAB1(config) #

**Configure Password on Cisco Router**

**Syntax:-**

Router(config)#line console 0

Router(config-line)#password PASSWORD

Router(config-line)#login

|  |  |
| --- | --- |
| **COMMAND** | **DESCRIPTION** |
| Router(config)#line console 0 | Move in console line mode |
| Router(config-line)#password | Set console line password to PASSWORD |
| Router(config-line)#login | Enable password authentication for console line |

**Configure clock time zone:-**

Router allows us to localize the time zone. Following command will set time zone to +5 hour of EST [Eastern Standard Time]

**Syntax:-**

Router(config)#clock timezone EST 05

**Configure Serial Interface in Router:-**

**Syntax:-**

Router(config) #interface serial 0/0/0

Router(config-if) #ip address [default\_gateway\_address] [subnet\_mask\_address]

Router(config-if) #clock rate TIME

Router(config-if) #bandwidth BANDWIDTH (in bps)

Router(config-if) #no shutdown

Router(config-if) #exit

|  |  |
| --- | --- |
| **COMMAND** | **DESCRIPTION** |
| Router(config)#interface serial 0/0/0 | Enter into serial interface 0/0/0 configuration mode |
| Router(config-if)#description | Optional command. It set description on interface that is locally significant |
| Router(config-if)#ip address 10.0.0.1 255.0.0.0 | Assigns address and subnet mask to interface |
| Router(config-if)#clock rate 64000 | DCE side only command. Assigns a clock rate for the interface |
| Router(config-if)#bandwidth 64 | DCE side only command. Set bandwidth for the interface. |
| Router(config-if)#no shutdown | Turns interface on |

**Configure Fastethernet Interface in Router:-**

**Syntax:-**

Router(config)#interface fastethernet 0/0

Router(config-if)#description Development department

Router(config-if)#ip address [default\_gateway\_address] [subnet\_mask\_address]

Router(config-if)#no shutdown

|  |  |
| --- | --- |
| **COMMAND** | **DESCRIPTION** |
| Router(config)#interface fastethernet 0/0 | Enter into the FastEthernet 0/0 interface. |
| Router(config-if)#description Development department | This command is optional. It will set description on interface. |
| Router(config-if)#ip address 192.168.0.1 255.255.255.0 | Assigns address and subnet mask to interface |
| Router(config-if)#no shutdown | Turns interface on. All interfaces are set to off on startup. |

**Static route command:-**

**Syntax:-**

Router# show ip route [[ipAddress [mask]] [bgp | connected | mpls [ipAddress] [ipAddress/PrefLen] [ipAddress mask] [detail] | isis | ospf | static | summary | multicast |unicast]

The following table describes the fields displayed by the show ip route command.

|  |  |
| --- | --- |
| **COMMAND** | **DESCRIPTION** |
| connected | Specifies the route was learned as a result of configuring the interface. |
| static | Specifies the route was explicitly configured using the ip route command. |
| BGP | Specifies the route was received from BGP protocol advertisements. |
| MPLS | Specifies this route carries MPLS data. |
| external type 1 | Specifies the route was imported into the OSPF area from an Autonomous System Boundary Router (ASBR). Further, this route used the cost associated with the link between the ASBR and this router as part of the cost of the route. |
| external type 2 | Specifies the route was imported into the OSPF area from an Autonomous System Boundary Router (ASBR). Further, this route does not use the cost associated with the link between the ASBR and this router as part of the cost of the route. |
| isis | Specifies this route was received from IS-IS protocol advertisements. |
| N.N.N.N/nn | Network address and mask of the remote network. |
| via N.N.N.N | Indicates the next router in the remote network. |
| directly connected to | Indicates the network is directly connected to a local interface. |
| UD | Indicates the Up/Down bit is set and that the route was leaked from L2 into L1. |
| d: | The administrative distance assigned to this route. |
| m: | The metric assigned to this route. |

Router(config)# ip route destination\_network\_#[subnet\_mask]IP\_address\_of\_next\_hop\_neighbor [administrative\_distance] [permanent]

1. **ip route**

It is the command that add new route in routing table.

1. **destination\_network\_#[subnet\_mask**

This is the first parameter. It specifies the destination network address. We need to provide subnet mask if we are using sub-network. If we are not using sub-network then we can omit the subnet mask value. It will parse automatically.

1. **IP\_address\_of\_next\_hop\_neighbor / interface\_to\_exit**

This parameter provides a way to reach the destination network. Both commands use separate way to assign this value. First command provides the IP address of next hop neighbor. It tells router that if it receives a packet for destination [that we set in previous parameter], forward that packet to this next hop neighbor IP address.Second command also do the same job but in different way. It specifies exit interface instead of next hop IP address. It tells router that if it receives a packet for the destination specified by previous parameter then exits that packet from this interface.

1. **administrative\_distance**

Administrative distance is the trustworthiness of route. Route with the lowest AD value will be chosen while forwarding the packet. By default static route has two AD values depending on the previous parameter. If you have used next hop neighbor IP address, then the default AD value will be **1**. If you have used exit interface, then the default AD value will be **0**. This parameter allows us to create multiple static routes for the same destination.

1. **permanent**

When a route goes down router will remove that from routing table. Permanent parameter will keep this route in routing table even if it goes down. Its optional parameter we can omit it. If we omit it, router will remove this route from routing table if it goes down

**Implement Rip:-**

**Syntax:-**

Router(config)# router rip

Router(config-router)#network IP\_address

|  |  |
| --- | --- |
| **Commands** | **Descriptions** |
| Router(config)#router rip | Enables RIP as a routing protocol |
| Router(config-router)#network w.x.y.z | w.x.y.z is the network number of the directly connected network you want to advertise. |
| Router(config)#no router rip | Turns off the RIP routing process |
| Router(config-router)#no network w.x.y.z | Removes network w.x.y.z from the RIP routing process. |
| Router(config-router)#version 2 | RIP will now send and receive RIPv2 packets globally. |
| Router(config-router)#version 1 | RIP will now send and receive RIPv1 packets only |
| Router(config-router)#no auto-summary | RIPv2 summarizes networks at the classful boundary. This command turns autosummarization off. |
| Router(config-router)#passive-interface s0/0/0 | RIP updates will not be sent out this interface. |
| Router(config-router)#no ip split-horizon | Turns off split horizon (on by default). |
| Router(config-router)#ip split-horizon | Re-enables split horizon |
| Router(config-router)#timers basic 30 90 180 270 360 | Changes timers in RIP: 30 = Update timer (in seconds) 90 = Invalid timer (in seconds) 180 = Hold-down timer (in seconds) 270 = Flush timer (in seconds) 360 = Sleep time (in milliseconds) |
| Router#debugip rip | Displays all RIP activity in real time |
| Router#showip rip database | Displays contents of the RIP database |

**Example:-**

Router(config)# router rip

Router(config-router)#network 10.0.0.0

Router(config-router)#network 20.0.0.0

Router(config-router) #exit

Router(config)#

**Implement OSPF:-**

**Syntax:-**

Router(config)# router ospfprocess\_ID

Router(config-router)#network IP\_addresswildcard\_mask area area\_#

|  |  |
| --- | --- |
| **Commands** | **Descriptions** |
| Router(config)#router ospf 1 | Starts OSPF process 1. The process ID is any positive integer value between 1 and 65,535. |
| Router(config-router)#network 172.16.0.0 0.0.255.255 area 0 | OSPF advertises interfaces, not networks. Uses the wildcard mask to determine which interfaces to advertise. |
| Router(config-if)#ipospfhellointerval timer 20 | Changes the Hello Interval timer to 20 seconds. |
| Router(config-if)#ipospfdeadinterval 80 | Changes the Dead Interval timer to 80 seconds. |
| Router#showip protocol | Displays parameters for all protocols running on the router |
| Router#showip route | Displays a complete IP routing table |
| Router#showipospf | Displays basic information about OSPF routing processes |
| Router#showipospf interface | Displays OSPF info as it relates to all interfaces |
| Router#showipospf interface fastethernet 0/0 | Displays OSPF information for interface fastethernet 0/0 |
| Router#showipospf border-routers | Displays border and boundary router information |
| Router#showipospfneighbor | Lists all OSPF neighbors and their states |
| Router#showipospfneighbor detail | Displays a detailed list of neighbors |
| Router#clearip route \* | Clears entire routing table, forcing it to rebuild |
| Router#clearip route a.b.c.d | Clears specific route to network a.b.c.d |
| Router#clearipopsf counters | Resets OSPF counters |
| Router#clearipospf process | Resets entire OSPF process, forcing OSPF to re-create neighbors, database, and routing table |
| Router#debugipospf events | Displays all OSPF events |
| Router#debugipospf adjacency | Displays various OSPF states and DR/ BDR election between adjacent routers |
| Router#debugipospf packets | Displays OPSF packets |

**Example:-**

Router(config)# router ospf 1

Router(config-router)#network 10.0.0.0 0.255.255.255 area 0

Router(config-router)#network 20.0.0.0 0.255.255.255 area 0

Router(config-router) #exit

Router(config)#

1. **PC Configuration:-**

Step 1: Select An End Device. (Eg. PC 0)  
Step 2: Clickon Pc 0 🡪 Desktop🡪 IP Configuration.

Step 3: AssignIp Address (Eg. 192.168.15.2)

Step 3: Assign Subnet Mask In This Case, 255.255.255.0

Step 4: Assign Default Gateway In This Case, 192.168.15.1

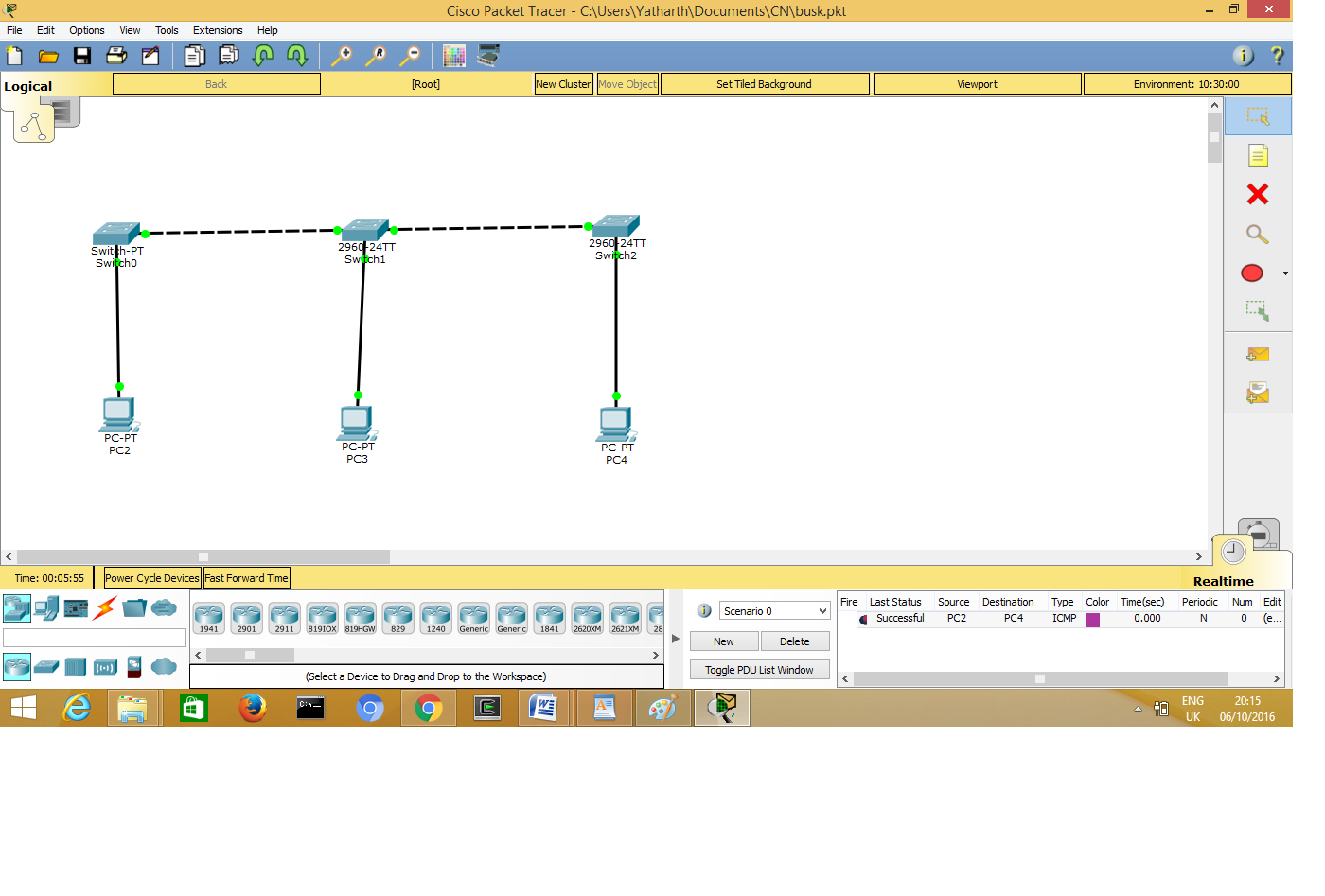
**CONCLUSION:** Thus the installation of Packet Tracer in Linux environment and simulation commands are studied.

**PRACTICAL NO. 7**

**AIM:** Simulation of BUS, Mesh , Ring and Star topology by using packet tracer tool.

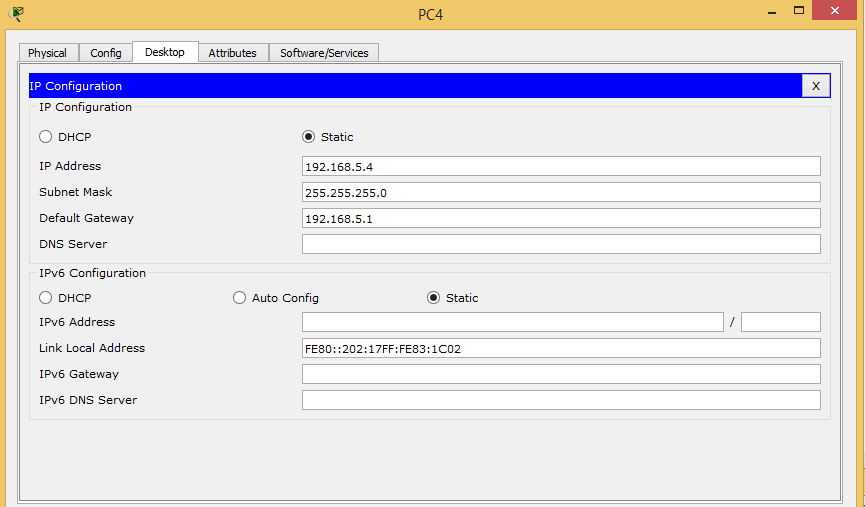
**OUTPUT:**

1. **BUS Topology**

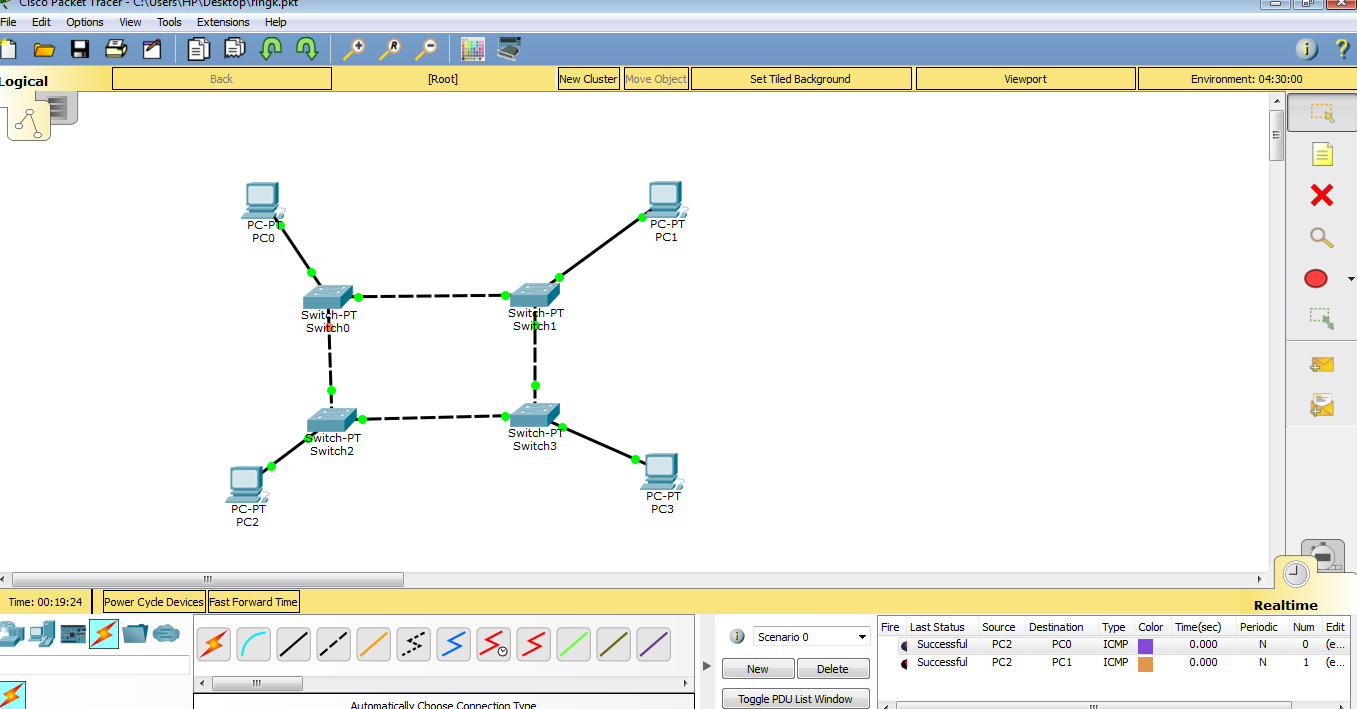


|  |  |
| --- | --- |
| **IP Configuration of PC2:** | **IP Configuration of PC3:** |
| busk_pc2.png | busk_pc3.png |

**IP Configuration of PC4:**



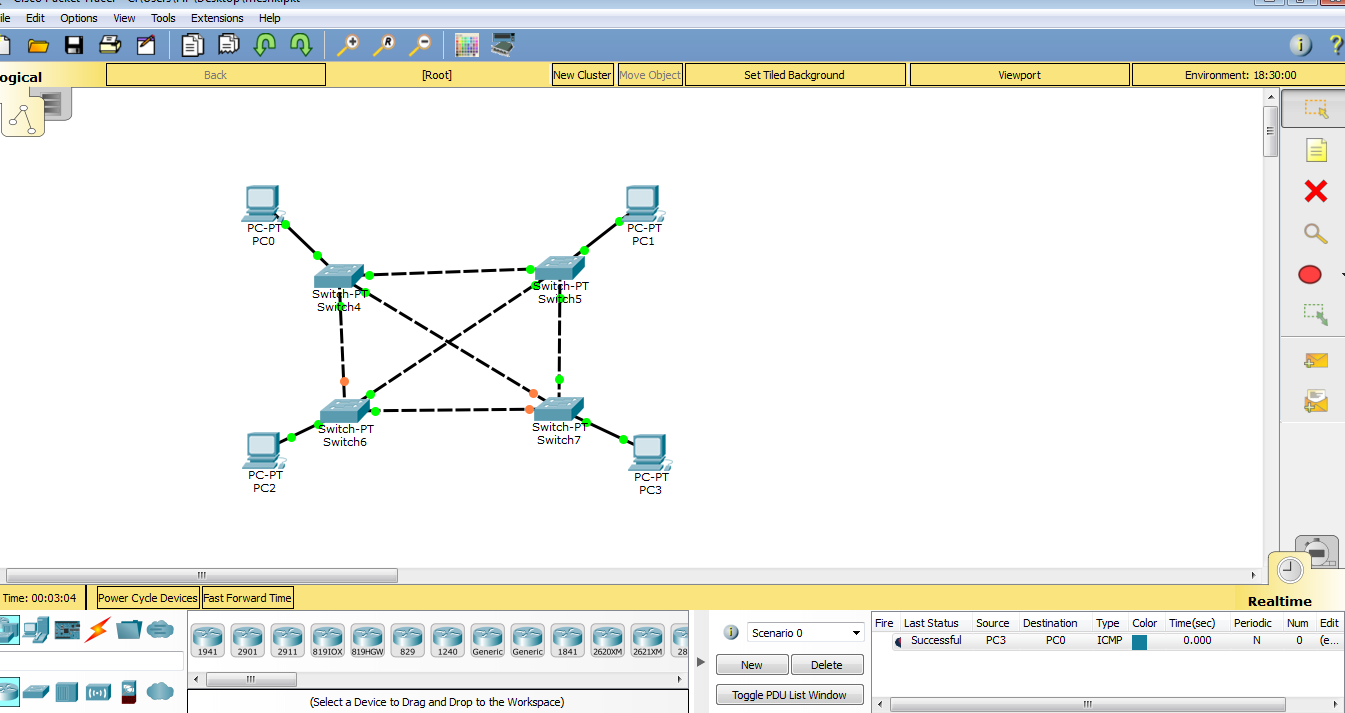
1. **Ring Topology:**



|  |  |
| --- | --- |
| **IP Configuration of PC0** | **IP Configuration of PC1** |
| ringk_pc0.png | ringk_pc1.png |

|  |  |
| --- | --- |
| **IP Configuration of PC2** | **IP Configuration of PC3** |
| ringk_pc2.png | ringk_pc3.png |

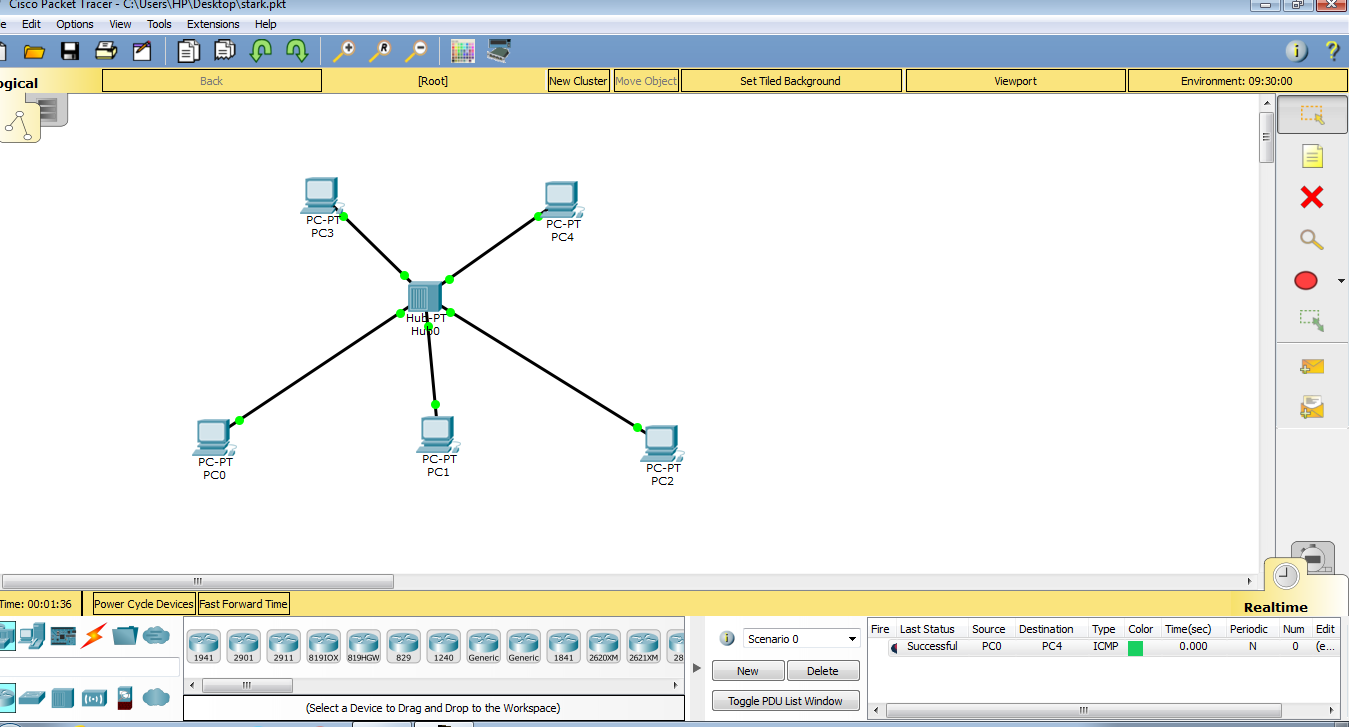
1. **Mesh Topology:**



|  |  |
| --- | --- |
| **IP Configuration of PC0** | **IP Configuration of PC1** |
| meshk_pco.png | meshk_pc1.png |

|  |  |
| --- | --- |
| **IP Configuration of PC2** | **IP Configuration of PC3** |
| ringk_pc2.png | meshk_pc3.png |

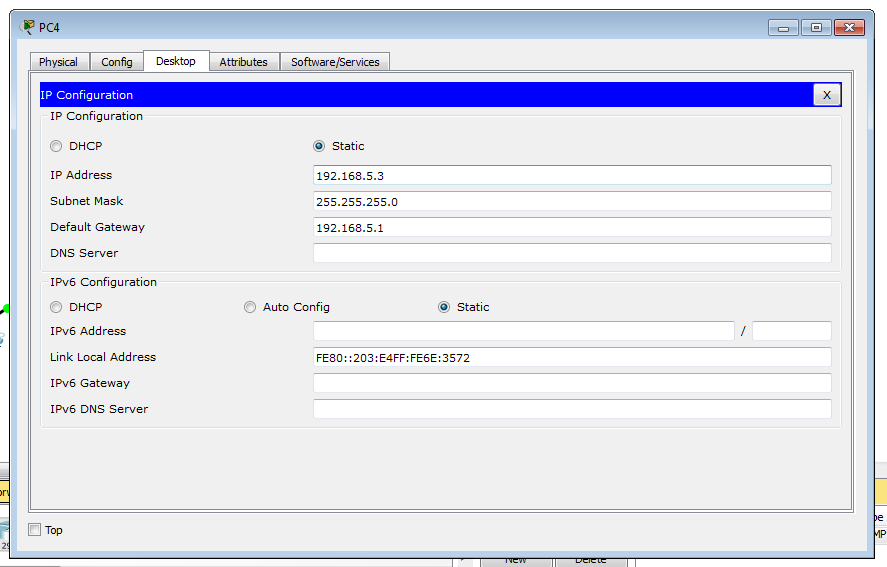
1. **Star Topology:**



|  |  |
| --- | --- |
| **IP Configuration of PC0:** | **IP Configuration of PC1:** |
| stark_pc0.png | stark_pc1.png |

|  |  |
| --- | --- |
| **IP Configuration of PC2:** | **IP Configuration of PC3:** |
| stark_pc2.png | stark_pc3.png |

**IP Configuration of PC4:**

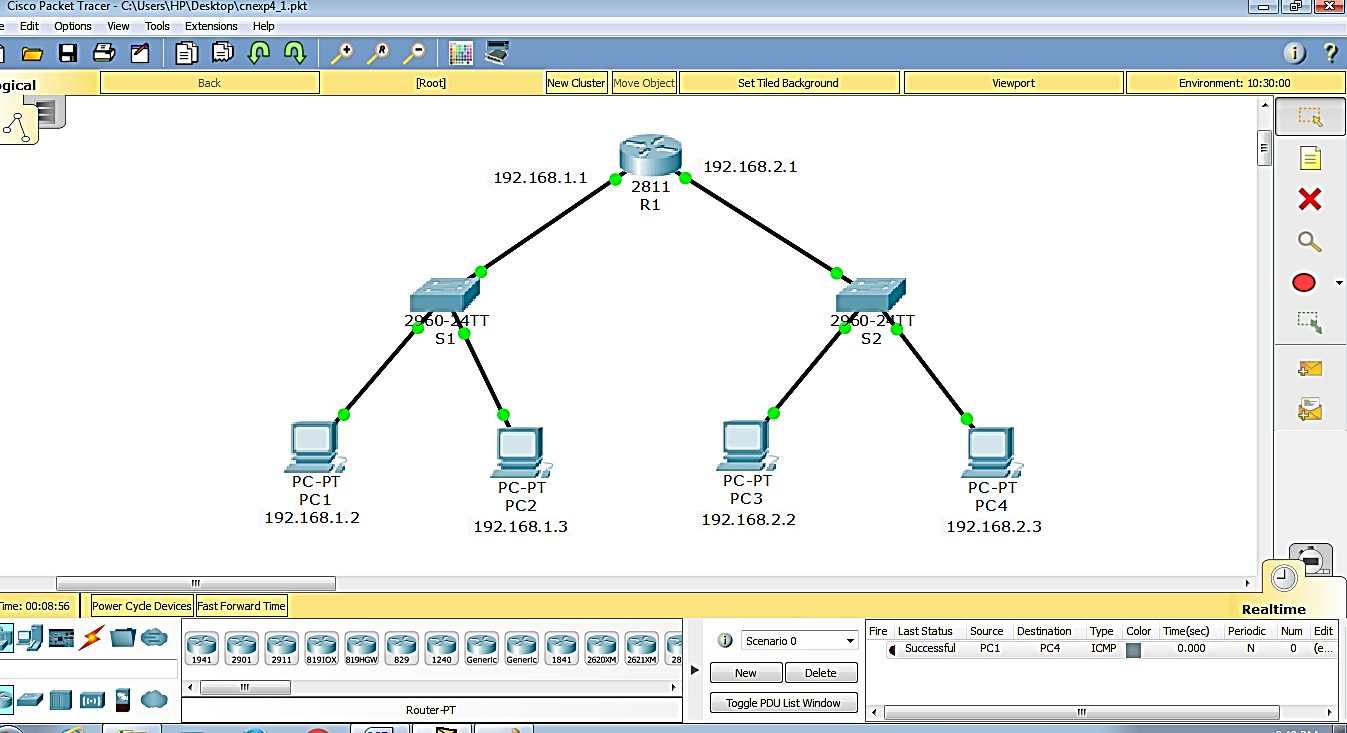


**CONCLUSION:** Thus BUS, Ring, Mesh and Star topologies have been implemented in packet tracer.

**PRACTICAL NO. 8**

**AIM:** Simulation of connection of end devices using router and switches (using packet tracer tool).

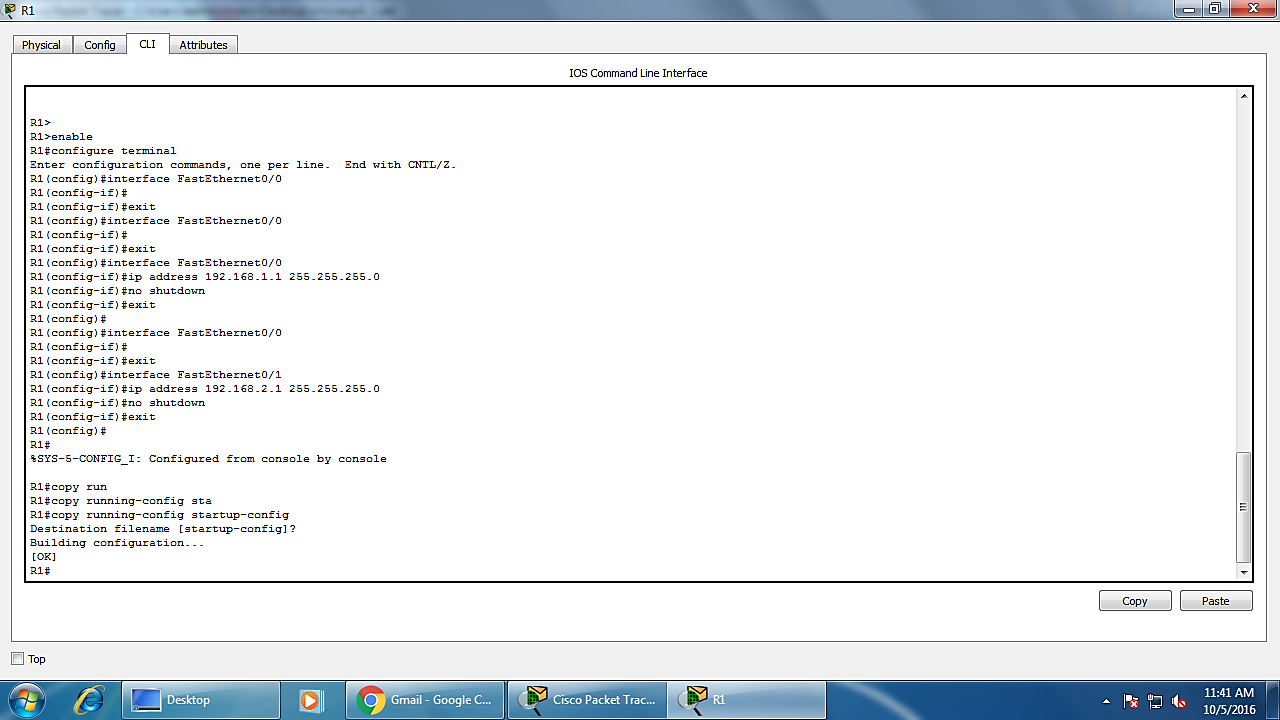
**OUTPUT:**



|  |  |
| --- | --- |
| **IP Configuration of PC1** | **IP Configuration of PC2** |
| cn4_pc1.png | cn4_pc2.png |

|  |  |
| --- | --- |
| **IP Configuration of PC3** | **IP Configuration of PC4** |
| cn4_pc3.png | cn4_pc4.png |

**Configuration of Router R1**

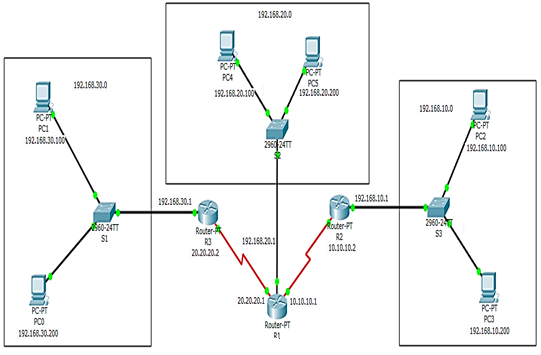


**CONCLUSION:** Thus a network is created using Routers and Switches in packet tracer

**PRACTICAL NO. 9**

**AIM:** Simulation of connection between 3 cities for using IP route command using static routing

**OUTPUT:**

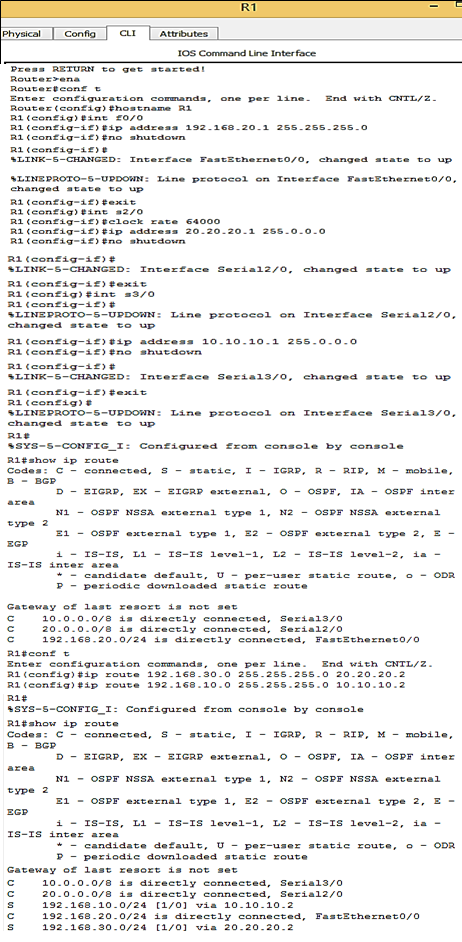


|  |  |
| --- | --- |
| **IP Configuration of PC0** | **IP Configuration of PC1** |
| cnexp4_2_pc0.png | cnexp4_2_pc1.png |

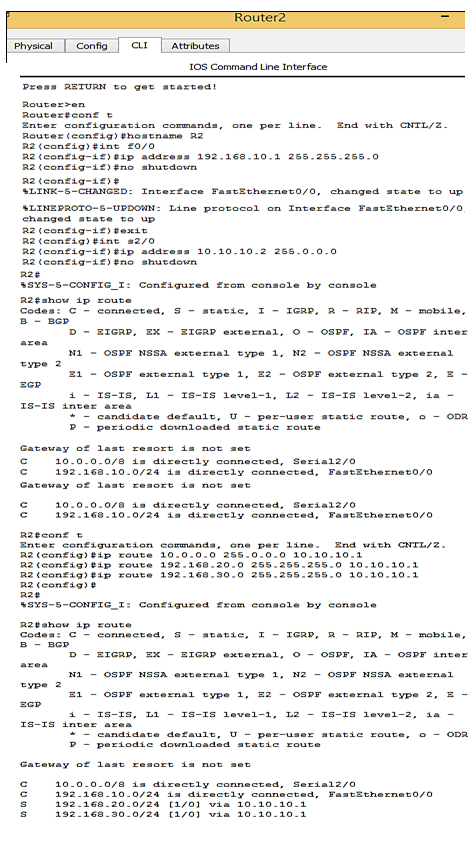
|  |  |
| --- | --- |
| **IP Configuration of PC2** | **IP Configuration of PC3** |
| cnexp4_2_pc2.png | cnexp4_2_pc3.png |

|  |  |
| --- | --- |
| **IP Configuration of PC4** | **IP Configuration of PC5** |
| cnexp4_2_pc4.png | cnexp4_2_pc5.png |

**Configuration of Router R1:**



**Configuration of Router R2**



**Configuration of Router R3**



**CONCLUSION:** Thus the connection of 3 cities have been successfully implemented using Packet tracer

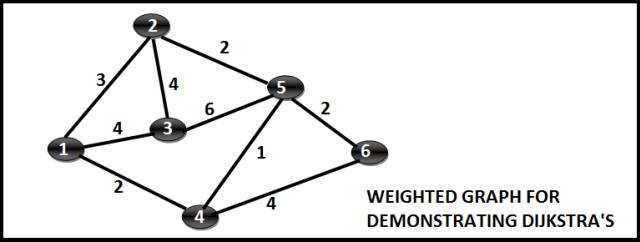
**Practical No.10**

**AIM:** Implementation of Shortest Path Algorithm (Dijkstra’s Algorithm) using C/C++/JAVA

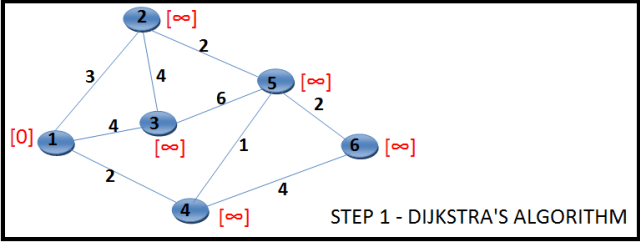
**Theory:**

The idea of the algorithm is very simple.

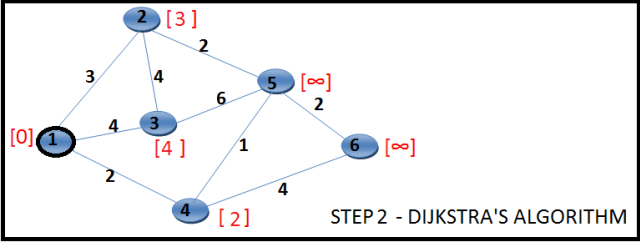
1. It maintains a list of unvisited vertices.
2. It chooses a vertex (the source) and assigns a maximum possible cost (i.e. infinity) to every other vertex.
3. The cost of the source remains zero as it actually takes nothing to reach from the source vertex to itself.
4. In every subsequent step of the algorithm it tries to improve(minimize) the cost for each vertex. Here the cost can be distance, money or time taken to reach that vertex from the source vertex. The minimization of cost is a multi-step process.
   1. For each unvisited neighbor (vertex 2, vertex 3, vertex 4) of the current vertex (vertex 1) calculate the new cost from the vertex (vertex 1).
   2. For e.g. the new cost of vertex 2 is calculated as the minimum of the two ( (existing cost of vertex 2) or (sum of cost of vertex 1 + the cost of edge from vertex 1 to vertex 2) )
5. When all the neighbors of the current node are considered, it marks the current node as visited and is removed from the unvisited list.
6. Select a vertex from the list of unvisited nodes (which has the smallest cost) and repeat step 4.
7. At the end there will be no possibilities to improve it further and then the algorithm ends
8. For demonstration we will consider the below graph:

[](https://i2.wp.com/techieme.in/wp-content/uploads/DJK_0.png)

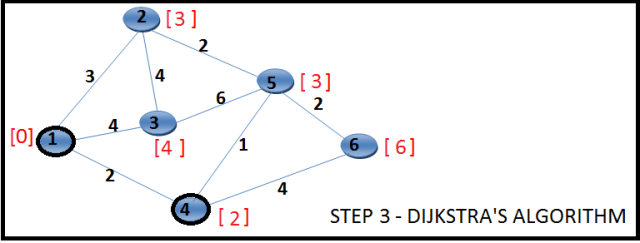
1. Step Wise Execution
2. Step 1:
3. Mark Vertex 1 as the source vertex. Assign a cost zero to Vertex 1 and (infinite to all other vertices). The state is as follows:

[](https://i1.wp.com/techieme.in/wp-content/uploads/DJK_1.png)

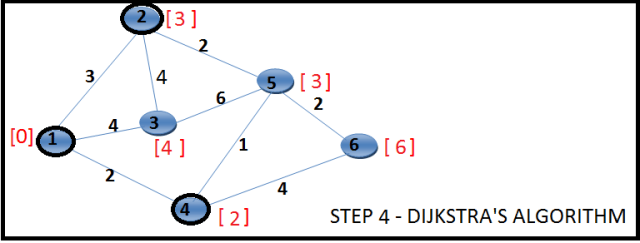
1. Step2**:**
2. For each of the unvisited neighbors (Vertex 2, Vertex 3 and Vertex 4) calculate the minimum cost as min(current cost of vertex under consideration, sum of cost of vertex 1 and connecting edge). Mark Vertex 1 as visited , in the diagram we border it black. The new state would be as follows:

[](https://i2.wp.com/techieme.in/wp-content/uploads/DJK_2.png)

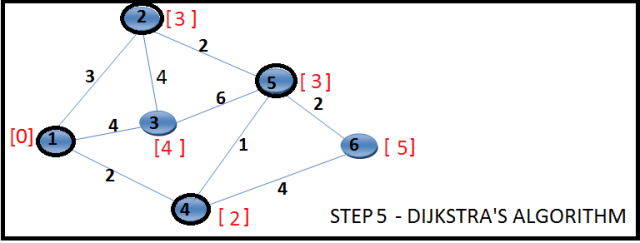
1. Step 3:
2. Choose the unvisited vertex with minimum cost (vertex 4) and consider all its unvisited neighbors (Vertex 5 and Vertex 6) and calculate the minimum cost for both of them. The state is as follows:

[](https://i0.wp.com/techieme.in/wp-content/uploads/DJK_3.png)

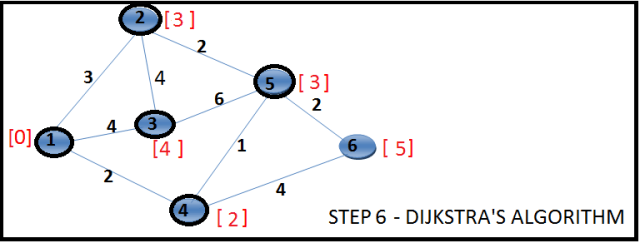
1. Step 4:
2. Choose the unvisited vertex with minimum cost (vertex 2 or vertex 5, here we choose vertex 2) and consider all its unvisited neighbors (Vertex 3 and Vertex 5) and calculate the minimum cost for both of them. Now, the current cost of Vertex 3 is [4] and the sum of (cost of Vertex 2 + cost of edge (2,3) ) is 3 + 4 = [7]. Minimum of 4, 7 is 4. Hence the cost of vertex 3 won’t change. By the same argument the cost of vertex 5 will not change. We just mark the vertex 2 as visited, all the costs remain same. The state is as follows:

[](https://i0.wp.com/techieme.in/wp-content/uploads/DJK_4.png)

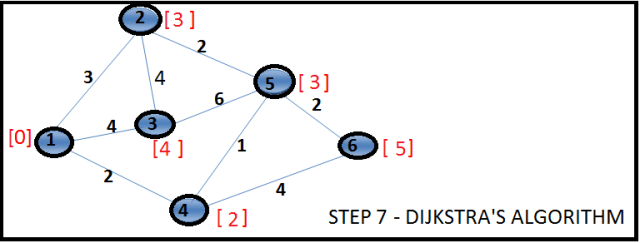
1. Step 5:
2. Choose the unvisited vertex with minimum cost (vertex 5) and consider all its unvisited neighbors (Vertex 3 and Vertex 6) and calculate the minimum cost for both of them. Now, the current cost of Vertex 3 is [4] and the sum of (cost of Vertex 5 + cost of edge (5,3) ) is 3 + 6 = [9]. Minimum of 4, 9 is 4. Hence the cost of vertex 3 won’t change. Now, the current cost of Vertex 6 is [6] and the sum of (cost of Vertex 5 + cost of edge (3,6) ) is 3 + 2 = [5]. Minimum of 6, 5 is 45. Hence the cost of vertex 6 changes to 5. The state is as follows:

[](https://i0.wp.com/techieme.in/wp-content/uploads/DJK_5.png)

1. Step 6:
2. Choose the unvisited vertex with minimum cost (vertex 3) and consider all its unvisited neighbors (none). So mark it visited. The state is as follows:

[](https://i1.wp.com/techieme.in/wp-content/uploads/DJK_6.png)

1. Step 7:
2. Choose the unvisited vertex with minimum cost (vertex 6) and consider all its unvisited neighbors (none). So mark it visited. The state is as follows:

[](https://i2.wp.com/techieme.in/wp-content/uploads/DJK_7.png)

1. Now there is no unvisited vertex left and the execution ends. At the end we know the shortest paths for all the vertices from the source vertex 1. Even if we know the shortest path length, we do not know the exact list of vertices which contributes to the shortest path until we maintain them separately or the data structure supports it.

**PROGRAM:**

import java.util.\*;

public class Dijkstra

{

public int distance[] = new int[10];

public int cost[][] = new int[10][10];

public void calc(int n,int s)

{

int flag[] = new int[n+1];

int i,min,pos=1,k,c,minimum;

for(i=1;i<=n;i++)

{

flag[i]=0;

this.distance[i]=this.cost[s][i];

}

c=2;

while(c<=n)

{

minimum=99;

for(k=1;k<=n;k++)

{

if(this.distance[k]<minimum && flag[k]!=1)

{

minimum=this.distance[i];

minpos=k;

}

}

flag[minpos]=1;

c++;

for(k=1;k<=n;k++)

{

if(this.distance[minpos]+this.cost[minpos][k] < this.distance[k] && flag[k]!=1 )

this.distance[k]=this.distance[minpos]+this.cost[minpos][k];

}

}

}

public static void main(String args[])

{

int nodes,source,i,j;

Scanner in = new Scanner(System.in);

System.out.println("Enter the Number of Nodes \n");

nodes = in.nextInt();

Dijkstra d = new Dijkstra();

System.out.println("Enter the Cost Matrix Weights: \n");

for(i=1;i<=nodes;i++)

for(j=1;j<=nodes;j++) {

d.cost[i][j]=in.nextInt();

if(d.cost[i][j]==0)

d.cost[i][j]=999;

}

System.out.println("Enter the Source Vertex :\n");

source=in.nextInt();

d.calc(nodes,source);

System.out.println("The Shortest Path from Source \t"+source+"\t to all other vertices are : \n");

for(i=1;i<=nodes;i++)

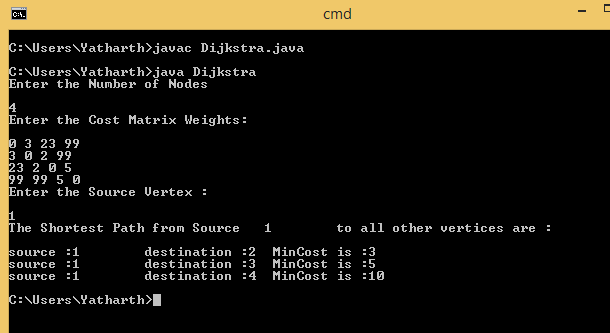
if(i!=source)

System.out.println("source :"+source+"\t destination :"+i+"\t MinCost is :"+d.distance[i]+"\t");

}

}

**OUTPUT:**



**CONCLUSION:** Thus we have implemented the shortest path algorithm( i.e Dijkstra’s algorithm) in JAVA

**Practical No.11**

**AIM:** Implementation of Distance Vector Algorithm (Bellmen Ford Algorithm) using C/C++/JAVA.

**Theory:**

The Bellman-Ford distance-vector routing algorithm is used by routers on internetworks to exchange routing information about the current status of the network and how to route packets to their destinations. The algorithm basically merges routing information provided by different routers into lookup tables. It is well defined and used on a number of popular networks. It also provides reasonable performance on small- to medium-sized networks, but on larger networks the algorithm is slow at calculating updates to the network topology. In some cases, looping occurs, in which a packet goes through the same node more than once. In general, most DVR (distance-vector routing) algorithms are not suitable for larger networks that have thousands of nodes, or if the network configuration changes often. In the latter case, the routing algorithm must be able to dynamically update the routing tables quickly to accommodate changes. A more efficient routing protocol is OSPF (Open Shortest Path First).

**PROGRAM:(Can be implemented in C,C++,Java)**

import java.util.Scanner;

public class BellmanFord {

private int distances[];

private int numberofvertices;

public static final int MAX\_VALUE = 999;

public BellmanFord(int numberofvertices) {

this.numberofvertices = numberofvertices;

distances = new int[numberofvertices + 1];

}

public void BellmanFordEvaluation(int source, int adjacencymatrix[][]) {

for (int node = 1; node <= numberofvertices; node++) {

distances[node] = MAX\_VALUE;

}

distances[source] = 0;

for (int node = 1; node <= numberofvertices - 1; node++) {

for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++)

{

for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++) {

if (adjacencymatrix[sourcenode][destinationnode] != MAX\_VALUE) {

if (distances[destinationnode] > distances[sourcenode]

+ adjacencymatrix[sourcenode][destinationnode])

distances[destinationnode] = distances[sourcenode]

+ adjacencymatrix[sourcenode][destinationnode];

}

}

}

}

for (int sourcenode = 1; sourcenode <= numberofvertices; sourcenode++) {

for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++) {

if (adjacencymatrix[sourcenode][destinationnode] != MAX\_VALUE)

{

if (distances[destinationnode] > distances[sourcenode]

+ adjacencymatrix[sourcenode][destinationnode])

System.out.println("The Graph contains negative egde cycle");

}

}

for (int vertex = 1; vertex <= numberofvertices; vertex++) {

System.out.println("distance of source"+source+"to"

+vertex+ "is" + distances[vertex]);

}

}

public static void main(String... arg) {

int numberofvertices = 0;

int source;

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of vertices");

numberofvertices = scanner.nextInt();

int adjacencymatrix[][] = new int[numberofvertices + 1][numberofvertices + 1];

System.out.println("Enter the adjacency matrix");

for (int sourcenode = 1; sourcenode<=numberofvertices; sourcenode++) {

for (int destinationnode = 1; destinationnode <= numberofvertices; destinationnode++) {

adjacencymatrix[sourcenode][destinationnode] = scanner.nextInt();

if (sourcenode == destinationnode) {

adjacencymatrix[sourcenode][destinationnode] = 0;

continue;

}

if (adjacencymatrix[sourcenode][destinationnode] == 0) {

adjacencymatrix[sourcenode][destinationnode] = MAX\_VALUE;

}

}

}

System.out.println("Enter the source vertex");

source = scanner.nextInt();

BellmanFord bellmanford = new BellmanFord(numberofvertices);

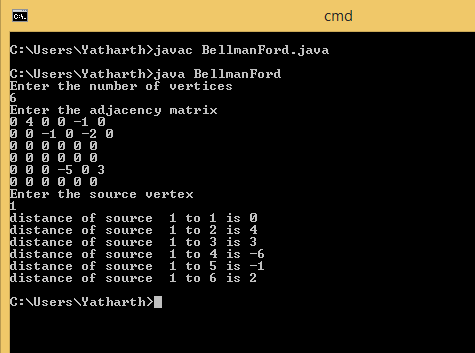
bellmanford.BellmanFordEvaluation(source, adjacencymatrix);

scanner.close();

}

}

**OUTPUT:**



**CONCLUSION:** Thus the Distance Vector algorithm(i.e Bellman Ford algorithm) is successfully implemented in JAVA

**PRACTICAL NO.12**

**AIM:** Implementation of Client/Server using TCP protocol

**Theory:**

TCP Connections Overview

To create a client/server relationship between systems, you must follow a particular set of conventions:

* Your systems must be connected with appropriate networking hardware and software, including TCP/IP protocol software.
* Systems communicate with each other through a TCP port. The processes at both ends of the connection must use the same port number.
* You specify either the TCP port number, or the device name of the device that represents it, as the device in Cache OPEN, USE, and CLOSE commands.

Using these conventions, the general procedure of establishing a TCP binding connection is:

1. The server process issues an OPEN command to a TCP device.
2. The server process issues a USE command, followed by a READ command, awaiting input from the client process. The server must be listening before a client can establish a connection. The initial READ command completes when the client has opened the connection and sent some data. You can include the “A” mode parameter in the OPEN command to make the initial READ complete as soon as the server accepts the connection.
3. The client process issues an OPEN command that specifies the TCP device to which it is connecting.
4. The client process issues a USE command followed by a WRITE command to complete the connection. Cache copies all characters in the WRITE command(s) to a buffer. It does not write them to the network until you issue a WRITE ! or WRITE # command to flush the buffer.
5. After the server has read the characters that the client sent in its first WRITE command, both sides can continue to issue READ and WRITE commands. There is no further restriction on the order of these commands to the same port.
6. Either side can initiate the closing of a connection with the CLOSE or HALT command. Closing the client side first is preferable. If the server needs to disconnect so that it can accept a connection from another client process, it can instead issue either a WRITE \*-2 command, or for compatibility with older versions, a USE command with the “DISCONNECT” option, and follow either one with a READ command.

**Program:**

**//tcpserver.java**

import java.io.\*;

import java.net.\*;

public class Tcpserver {

public static void main(String a[]) throws Exception {

System.out.println("TCP SERVER");

System.out.println("Server is ready to connect…");

ServerSocket serversoc=new ServerSocket(9);

Socket clientsoc = serversoc.accept();

PrintWriter out = new PrintWriter(clientsoc.getOutputStream(), true);

BufferedReader in = new BufferedReader(new

InputStreamReader(clientsoc.getInputStream()));

String inputline;

BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

try {

while (true) {

inputline = stdin.readLine();

out.println(inputline);

System.out.println("Client Says : "+in.readLine());

}

}

catch(Exception e) {

System.exit(0);

}

}

}

**//tcpclient.java**

import java.io.\*;

import java.net.\*;

public class tcpclient {

public static void main(String[] args) throws IOException {

System.out.println("TCP CLIENT");

System.out.println("Enter the host name to connect");

DataInputStream inp=new DataInputStream(System.in);

String str=inp.readLine();

Socket clientsoc = new Socket(str, 9);

PrintWriter out = new PrintWriter(clientsoc.getOutputStream(), true);

BufferedReader in = new BufferedReader(new

InputStreamReader(clientsoc.getInputStream()));

BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));

String userinput;

try {

while (true)

{

System.out.println("Sever Says : " + in.readLine());

userinput = stdin.readLine();

out.println(userinput);

}

}

catch(Exception e) {

System.exit(0);

}

}

}

**Conclusion**: Thus we have implemented the client server program using TCP protocol in JAVA