SRI SHANMUGHA COLLEGE OF ENGINEERING AND TECHNOLOGY

(APPROVED BY AICTE, NEW DELHI & AFFILIATED TO ANNA UNIVERSITYAND ACCREDITED BY NAAC & NBA(ECE,CSE,MECH)

Tiruchengode-Sankari Main Road, Pullipalayam, Morur(Po),
Sankari (Tk), Salem (Dt) Pin: 637 304

RECORD NOTE BOOK

(CS8581)NETWORKS LABORATORY

NAME :

REG NO :

YEAR :

SRI SHANMUGHA COLLEGE OF ENGINEERING AND TECHNOLOGY

(APPROVED BY AICTE, NEW DELHI & AFFILIATED TO ANNA UNIVERSITYAND ACCREDITED BY NAAC & NBA(ECE,CSE,MECH)

Tiruchengode-Sankari Main Road, Pullipalayam, Morur(Po),

Sankari (Tk), Salem (Dt) Pin: 637 304

RECORD NOTE BOOK

REG NO.

Certified that this is a bonafide record of Practical work done by

Mr/Ms …………………………………………. of the ……………………

Semester ……………….. Branch during the Academic year …………

in the ……………………………………………. Laboratory

Staff-in-charge Head of the Department

Submitted for the Anna University Practical Examination held on……………..

Internal Examiner External Examiner

LIST OF EXPERIMENTS

Ex.No	Date	Name Of The Experiment	Page No	Staff Signature	Remarks
1		Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute.Capture ping and traceroute PDUs using a network protocol analyzer and examine.			
2		Write a HTTP web client program to download a web page using TCP sockets			
3		Applications using CP sockets like: Echo client and echo server, Chat and File Transfer			
4		Simulation of DNS using UDP Sockets			
5		Write a code simulating ARP /RARP protocols			
6		Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS			
7		Study of TCP/UDP performance using Simulation tool.			
8		Simulation of Distance Vector/ Link State Routing algorithm.			
9		Performance Evaluation of Routing protocols using Simulation tool. i)UNICAST ROUTING PROTOCOL ii) MULTICASTING ROUTING PROTOCOL			
10		Simulation of ErrorDetection Code (like CRC)			
11		TOPIC BEYOND SYLLABUS i) Simulation of Go Back N protocol ii) CARRIER SENSE MULTIPLE ACCESS			

Ex. No : 1 Date :	**Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute.Capture ping and traceroute PDUs using a network protocol analyzer and examine.**

AIM: To Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute ping.

PRE LAB DISCUSSION:

Tcpdump:

The tcpdump utility allows you to capture packets that flow within your network to assist in network troubleshooting. The following are several examples of using tcpdump with different options. Traffic is captured based on a specified filter.

Netstat

Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX and other operating systems.

Netstat provides information and statistics about protocols in use and current TCP/IP network connections.

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.

From the command prompt, type **ipconfig** to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapter.

nslookup

The **nslookup** (which stands for *name server lookup*) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System.

Trace route:

Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Traceroute also records the time taken for each hop the packet makes during its route to the destination

Commands:

Tcpdump:

Display traffic between 2 hosts:

To display all traffic between two hosts (represented by variables host1 and host2): # tcpdump host host1 and host2

Display traffic from a source or destination host only:

To display traffic from only a source (src) or destination (dst) host:

tcpdump src host

tcpdump dst host

Display traffic for a specific protocol

Provide the protocol as an argument to display only traffic for a specific protocol, for example tcp, udp, icmp, arp

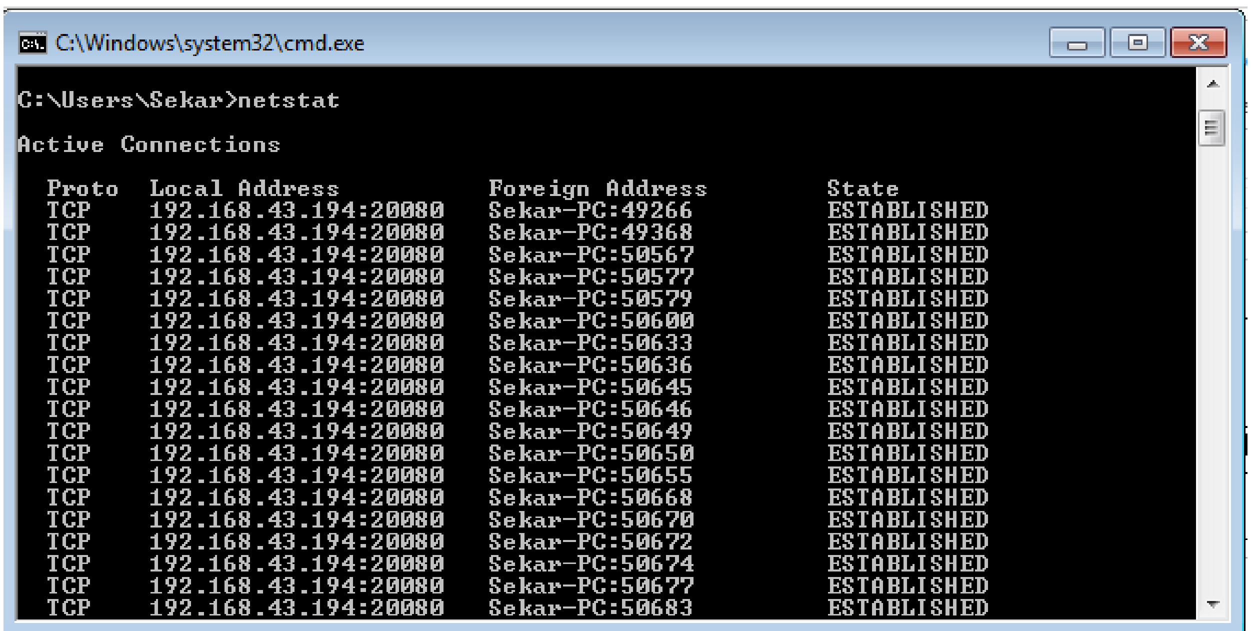
tcpdump protocol
For example to display traffic only for the tcp traffic :
tcpdump tcp
Filtering based on source or destination port
To filter based on a source or destination port:
tcpdump src port ftp
tcpdump dst port http

2.Netstat

Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX and other operating systems.

Netstat provides information and statistics about protocols in use and current TCP/IP network connections. The Windows help screen (analogous to a Linux or UNIX for netstat reads as follows: displays protocol statistics and current TCP/IP network connections.

#netstat



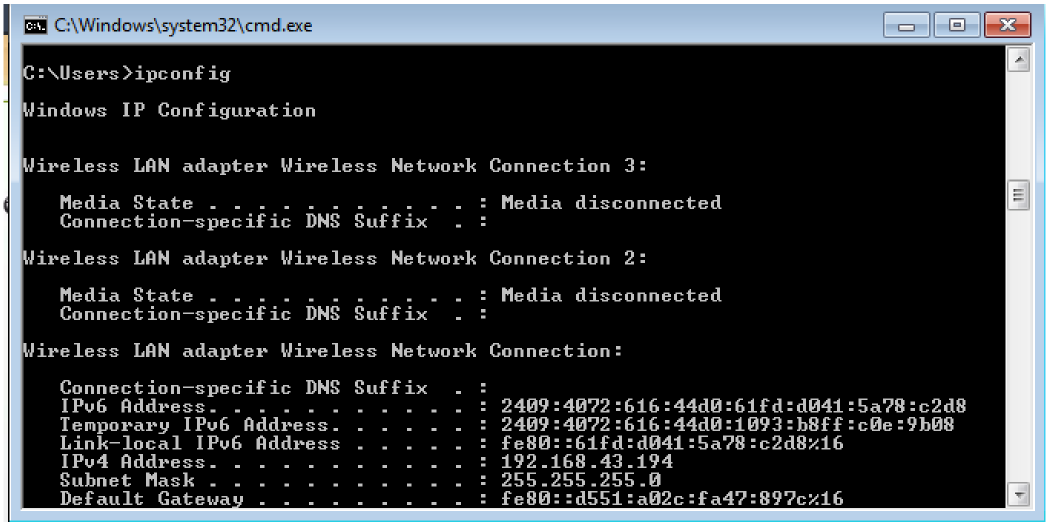
3. ipconfig

In Windows, **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.

Using ipconfig

From the command prompt, type **ipconfig** to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapter.

#ipconfig



4.nslookup

The **nslookup** (which stands for *name server lookup*) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System.

The nslookup command is a powerful tool for diagnosing DNS problems. You know you're experiencing a DNS problem when you can access a resource by specifying its IP address but not its DNS name.

#nslookup

5.Trace route:

Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values. The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop.
Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Traceroute also records the time taken for each hop the packet makes during its route to the destination. Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values.

The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop. Traceroute sends packets with TTL values that gradually increase from packet to packet, starting with TTL value of one. Routers decrement TTL values of packets by one when routing and discard packets whose TTL value has reached zero, returning the ICMP error message ICMP Time Exceeded.

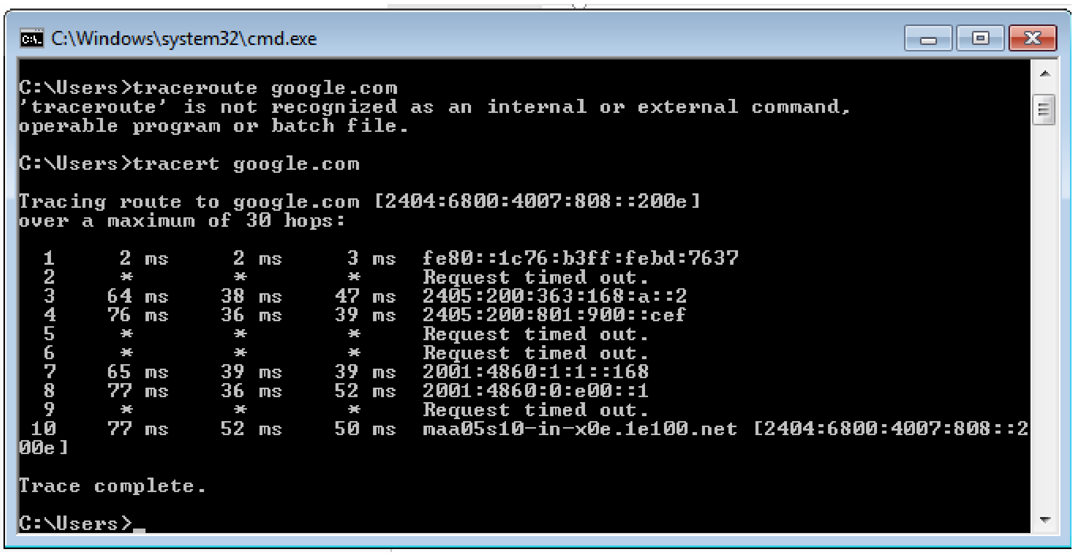
For the first set of packets, the first router receives the packet, decrements the TTL value and drops the packet because it then has TTL value zero. The router sends an ICMP Time Exceeded message back to the source. The next set of packets are given a TTL value of two, so the first router forwards the packets, but the second router drops them and replies with ICMP Time Exceeded.

Proceeding in this way, traceroute uses the returned ICMP Time Exceeded messages to build a list of routers that packets traverse, until the destination is reached and returns an ICMP Echo Reply message.

With the tracert command shown above, we're asking tracert to show us the path from the local computer all the way to the network device with the hostname

www.google.com.

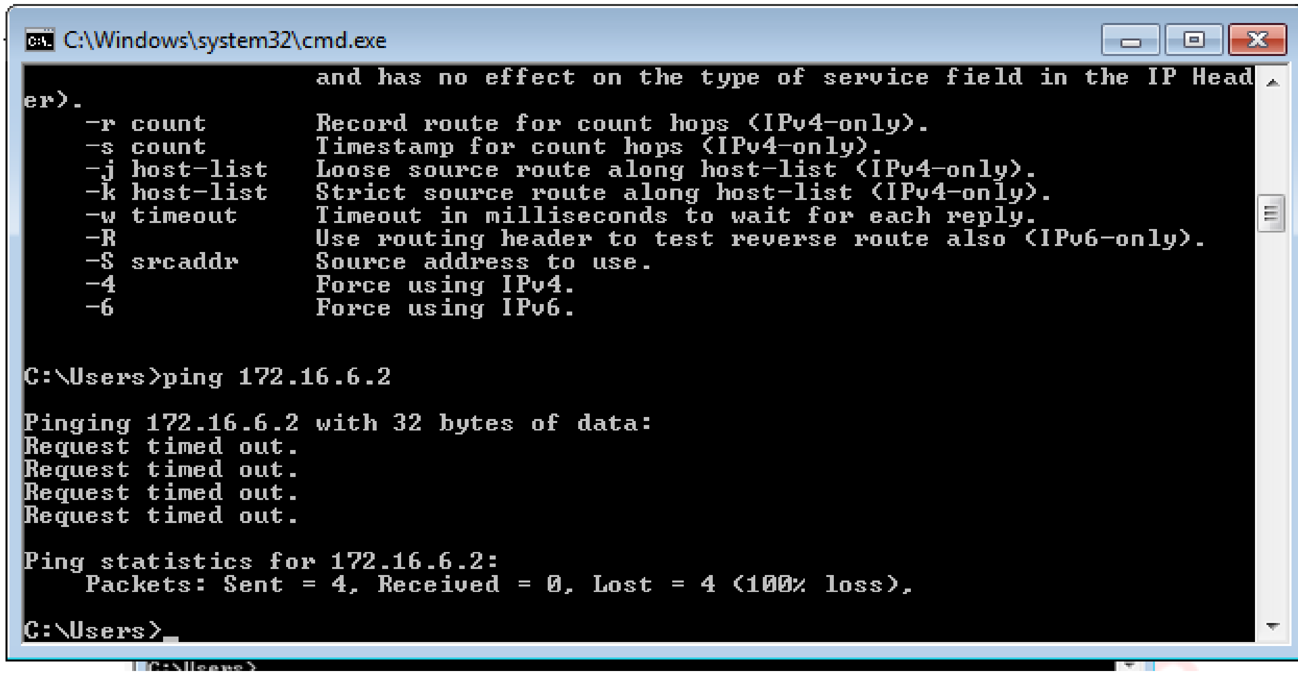
#tracert google.com



6. Ping:

The ping command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not. Tracking and isolating hardware and software problems. Determining the status of the network and various foreign hosts. The ping command is usually used as a simple way to verify that a computer can communicate over the network with another computer or network device. The ping command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to the destination computer and waiting for a response

ping172.16.6.2



RESULT:

Thus the various networks commands like tcpdump, netstat, ifconfig, nslookup and traceroute ping are executed successfully.

Ex. No : 2 Date :	**Write a HTTP web client program to download a web page using TCP sockets**

AIM:

To write a java program for socket for HTTP for web page upload and download .

PRE LAB DISCUSSION:

- HTTP means **H**yper**T**ext **T**ransfer **P**rotocol. HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
- For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page.
- The other main standard that controls how the World Wide Web works is HTML, which covers how Web pages are formatted and displayed. HTTP functions as a request–response protocol in the client–server computing model.
- A web browser, for example, may be the client and an application running on a computer hosting a website may be the server.
- The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs other functions on behalf of the client, returns a responsemessage to the client.
- The response contains completion status information about the request and may also contain requested content in its message body.

ALGORITHM:

Client:

1. Start.
2. Create socket and establish the connection with the server.
3. Read the image to be uploaded from the disk
4. Send the image read to the server
5. Terminate the connection
6. Stop.

Server:

1. Start
2. Create socket, bind IP address and port number with the created socket and make server a listening server.
3. Accept the connection request from the client
4. Receive the image sent by the client.
5. Display the image.
6. Close the connection.
7. Stop.

PROGRAM

Client

```
import javax.swing.*;
import  java.net.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.awt.image.BufferedImage; import
java.io.ByteArrayOutputStream; import
java.io.File;
import java.io.IOException; import
javax.imageio.ImageIO;
public class Client
{
        public static void main(String args[]) throws Exception
        {
                Socket  soc;
                BufferedImage img = null;
                soc=new
            Socket("localhost",4000);
                System.out.println("Client is running.
        ");

                try {
                        System.out.println("Reading image from disk. ");
                        img = ImageIO.read(new File("digital_image_processing.jpg"));
                        ByteArrayOutputStream baos = new ByteArrayOutputStream();
                        ImageIO.write(img, "jpg", baos);
                        baos.flush();
                        byte[] bytes = baos.toByteArray(); baos.close();
                        System.out.println("Sending image to server.");
                        OutputStream out = soc.getOutputStream();
                        DataOutputStream dos = new DataOutputStream(out);
                        dos.writeInt(bytes.length);
                        dos.write(bytes, 0, bytes.length);
                        System.out.println("Image sent to server. ");
                        dos.close();
                        out.close();
                }
                catch (Exception e)
                {
                        System.out.println("Exception: " + e.getMessage());
```

```
                soc.close();
            }
            soc.close();
        }
}
```

Server

```
import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;
class Server
{
        public static void main(String args[]) throws Exception
        {
            ServerSocket server=null;
            Socket socket;
            server=new ServerSocket(4000);
            System.out.println("Server Waiting for image");
            socket=server.accept(); System.out.println("Client connected.");
            InputStream in =    socket.getInputStream();
            DataInputStream dis = new DataInputStream(in);
            int len = dis.readInt();
            System.out.println("Image Size: " + len/1024 + "KB"); byte[] data = new byte[len];
            dis.readFully(data);
            dis.close();
            in.close();
            InputStream ian = new ByteArrayInputStream(data);
            BufferedImage bImage = ImageIO.read(ian);
            JFrame f = new JFrame("Server");
            ImageIcon icon = new ImageIcon(bImage);
            JLabel l = new JLabel();
            l.setIcon(icon);
            f.add(l);
            f.pack();
            f.setVisible(true);
        }
}
```

OUTPUT:

When you run the client code, following output screen would appear on client side.

.

RESULT:

Thus the socket program for HTTP for web page upload and download was developed and executed successfully.

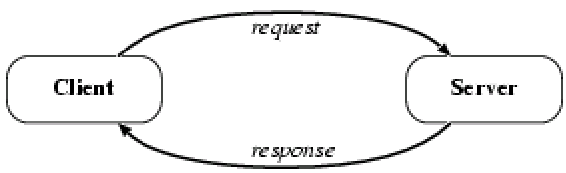
Ex. No : 3 Date :	**Applications using TCP sockets like: Echo client and echo server,Chat and File Transfer**

AIM

To write a java program for application using TCP Sockets Links

PRE LAB DISCUSSION:

- In the TCP Echo client a socket is created. Using the socket a connection is made to the server using the connect() function. After a connection is established, we send messages input from the user and display the data received from the server using send() and read() functions.
- In the TCP Echo server, we create a socket and bind to a advertized port number. After binding the process listens for incoming connections. Then an infinite loop is started to process the client requests for connections. After a connection is requested, it accepts the connection from the client machine and forks a new process.
- The new process receives data from the lient using recv() function and echoes the same data using the send() function. Please note hat this server is capable of handling multiple clients as it forks a new process for every client trying to connect to the server. TCP socket routines enable reliable IP communication using the transmission control protocol (TCP).
- The implementation of the Transmission Control Protocol (TCP) in the Network Component. TCP runs on top of the Internet Protocol (IP). TCP is a connection-oriented and reliable, full duplex protocol supporting a pair of byte streams, one for each direction.
- A TCP connection must be established before exchanging data. TCP retransmits data that do not reach the final destination due to errors or data corruption. Data is delivered in the sequence of its transmission



a.Echo client and echo server

ALGORITHM

Client

1. Start
2. Create the TCP socket
3. Establish connection with the server
4. Get the message to be echoed from the user

5. Send the message to the server
6. Receive the message echoed by the server
7. Display the message received from the server
8. Terminate the connection
9. Stop

Server

1. Start
2. Create TCP socket, make it a listening socket
3. Accept the connection request sent by the client for connection establishment
4. Receive the message sent by the client
5. Display the received message
6. Send the received message to the client from which it receives
7. Close the connection when client initiates termination and server becomes a listening server, waiting for clients.
8. Stop.

PROGRAM:

EchoServer.java

```
import java.net.*;
import java.io.*;
public class EServer
{
	public static void main(String args[])
	{
		ServerSocket s=null;
		String line;
		DataInputStream is;
		PrintStream ps;
		Socket c=null;
		try
		{
			s=new ServerSocket(9000);
		}
		catch(IOException e)
		{
			System.out.println(e);
		}
		try
		{
			c=s.accept();
			is=new DataInputStream(c.getInputStream());
```

```
                ps=new PrintStream(c.getOutputStream());
                while(true)
                {
                    line=is.readLine();
                    ps.println(line);
                }
            }
            catch(IOException e)
            {
                System.out.println(e);
            }
        }
}
```

EClient.java

```
import java.net.*;
import java.io.*;
public class EClient
{       public static void main(String arg[])
        {
            Socket c=null;
            String line;
            DataInputStream is,is1;
            PrintStream os;
            try
            {
                InetAddress ia = InetAddress.getLocalHost();
                c=new Socket(ia,9000);
            }
            catch(IOException e)
            {
                System.out.println(e);
            }
            try
            {
                os=new PrintStream(c.getOutputStream());
                is=new DataInputStream(System.in);
                is1=new DataInputStream(c.getInputStream());
                while(true)
                {
                System.out.println("Client:");
                    line=is.readLine();
```

```
                    os.println(line);
                    System.out.println("Server:" + is1.readLine());
                }
            }
            catch(IOException e)
            {
                System.out.println("Socket Closed!");
            }
        }}
```

OUTPUT

Server

C:\Program Files\Java\jdk1.5.0\bin>javac EServer.java
C:\Program Files\Java\jdk1.5.0\bin>java EServer
C:\Program Files\Java\jdk1.5.0\bin>

Client

C:\Program Files\Java\jdk1.5.0\bin>javac EClient.java
C:\Program Files\Java\jdk1.5.0\bin>java EClient
Client: Hai Server
Server:Hai Server
Client: Hello
Server:Hello
Client:end
Server:end
Client:ds
Socket Closed!

B.Chat

ALGORITHM

Client

1. Start
2. Create the UDP datagram socket
3. Get the request message to be sent from the user
4. Send the request message to the server
5. If the request message is “END” go to step 10
6. Wait for the reply message from the server
7. Receive the reply message sent by the server
8. Display the reply message received from the server
9. Repeat the steps from 3 to 8
10. Stop

Server

1. Start
2. Create UDP datagram socket, make it a listening socket
3. Receive the request message sent by the client
4. If the received message is “END” go to step 10
5. Retrieve the client’s IP address from the request message received
6. Display the received message
7. Get the reply message from the user
8. Send the reply message to the client
9. Repeat the steps from 3 to 8.
10. Stop.

PROGRAM

UDPserver.java

```
import java.io.*;
import java.net.*;
class UDPserver
{
	public static DatagramSocket ds;
	public static byte buffer[]=new byte[1024];
	public static int clientport=789,serverport=790;
	public static void main(String args[])throws Exception
	{
		ds=new DatagramSocket(clientport);
		System.out.println("press ctrl+c to quit the program");
		BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));
		InetAddress ia=InetAddress.geyLocalHost();
		while(true)
		{
			DatagramPacket p=new DatagramPacket(buffer,buffer.length);
			ds.receive(p);
			String psx=new String(p.getData(),0,p.getLength());
			System.out.println("Client:" + psx);
			System.out.println("Server:");
			String str=dis.readLine();
			if(str.equals("end"))
				break;
			buffer=str.getBytes();
			ds.send(new DatagramPacket(buffer,str.length(),ia,serverport));
		}
	}
}
```

UDPclient.java

```
import java .io.*;
import java.net.*;
class UDPclient
{
	public static DatagramSocket ds;
	public static int clientport=789,serverport=790;
	public static void main(String args[])throws Exception
	{
		byte buffer[]=new byte[1024];
		ds=new DatagramSocket(serverport);
		BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));
		System.out.println("server waiting");
		InetAddress ia=InetAddress.getLocalHost();
		while(true)
		{
			System.out.println("Client:");
			String str=dis.readLine();
			if(str.equals("end"))
				break;
			buffer=str.getBytes();
			ds.send(new DatagramPacket(buffer,str.length(),ia,clientport));
			DatagramPacket p=new DatagramPacket(buffer,buffer.length);
			ds.receive(p);
			String psx=new String(p.getData(),0,p.getLength());
			System.out.println("Server:" + psx);
		}
	}
}
```

OUTPUT:

Server

```
C:\Program Files\Java\jdk1.5.0\bin>javac UDPserver.java
C:\Program Files\Java\jdk1.5.0\bin>java UDPserver
press ctrl+c to quit the program
Client:Hai Server
Server:Hello Client
Client:How are You
Server:I am Fine
```

Client

```
C:\Program Files\Java\jdk1.5.0\bin>javac UDPclient.java
C:\Program Files\Java\jdk1.5.0\bin>java UDPclient
```

server waiting
Client:Hai Server
Server:Hello Clie
Client:How are You
Server:I am Fine
Client:end

C. File Transfer

AIM:

To write a java program for file transfer using TCP Sockets.

Algorithm

Server

1. Import java packages and create class file server.
2. Create a new server socket and bind it to the port.
3. Accept the client connection
4. Get the file name and stored into the BufferedReader.
5. Create a new object class file and realine.
6. If file is exists then FileReader read the content until EOF is reached.
7. Stop the program.

Client

1. Import java packages and create class file server.
2. Create a new server socket and bind it to the port.
3. Now connection is established.
4. The object of a BufferReader class is used for storing data content which has been retrieved from socket object.
5. The connection is closed.
6. Stop the program.

PROGRAM

File Server :

```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket
public class FileServer
{
    public static void main(String[] args) throws Exception
        {
            //Initialize Sockets
            ServerSocket ssock = new ServerSocket(5000); Socket
            socket = ssock.accept();
          //The InetAddress specification
              InetAddress IA = InetAddress.getByName("localhost");

        //Specify the file
            File file = new File("e:\\Bookmarks.html");
            FileInputStream fis = new
            FileInputStream(file);
              BufferedInputStream bis = new BufferedInputStream(fis); //Get
              socket's output stream
              OutputStream os = socket.getOutputStream(); //Read
            File Contents into contents array
              byte[] contents;
             long fileLength = file.length();
             long current = 0;
             long start = System.nanoTime();
             while(current!=fileLength){
                    int size = 10000;
                    if(fileLength - current >= size)
                      current += size;
                    else{
                            size = (int)(fileLength - current);
                            current = fileLength;
                     }
                contents = new byte[size];
```

```
                    bis.read(contents, 0, size);
                     os.write(contents);
                            System.out.print("Sending file ... "+(current*100)/fileLength+"% complete!");
                    }
                os.flush();
         //File transfer done. Close the socket connection!
         socket.close();
         ssock.close();
         System.out.println("File sent succesfully!");
         } }
```

File Client:

```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.Socket;

public class FileClient {
   public static void main(String[] args) throws Exception{
      //Initialize socket
            Socket socket = new Socket(InetAddress.getByName("localhost"), 5000); byte[]
              contents = new byte[10000];
      //Initialize the FileOutputStream to the output file's full path. FileOutputStream fos = new
            FileOutputStream("e:\\Bookmarks1.html");
               BufferedOutputStream bos = new
              BufferedOutputStream(fos);InputStream is =
              socket.getInputStream();
       //No of bytes read in one read() call
            int bytesRead = 0;
           while((bytesRead=is.read(contents))!=-1)
                 bos.write(contents, 0, bytesRead);
           bos.flush();
           socket.close();
           System.out.println("File saved successfully!");
   }
}
```

Output

server

E:\nwlab>java FileServer
Sending file ... 9% complete!
Sending file ... 19% complete!
Sending file ... 28% complete!
Sending file ... 38% complete!
Sending file ... 47% complete!
Sending file ... 57% complete!
Sending file ... 66% complete!
Sending file ... 76% complete!
Sending file ... 86% complete!
Sending file ... 95% complete!
Sending file ... 100% complete!
File sent successfully!

E:\nwlab>**client**
E:\nwlab>java FileClient
File saved successfully!

E:\nwlab>

RESULT:

Thus the java application program using TCP Sockets was developed and executed successfully.

Ex. No : 4 Date :	**Simulation of DNS using UDP Sockets**

AIM

To write a java program for DNS application

PRE LAB DISCUSSION:

- The Domain Name System (DNS) is a hierarchical decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities.
- The domain name space refers a hierarchy in the internet naming structure. This hierarchy has multiple levels (from 0 to 127), with a root at the top. The following diagram shows the domain name space hierarchy.
- Name server contains the DNS database. This database comprises of various names and their corresponding IP addresses. Since it is not possible for a single server to maintain entire DNS database, therefore, the information is distributed among many DNS servers.
- Types of Name Servers
- Root Server is the top level server which consists of the entire DNS tree. It does not contain the information about domains but delegates the authority to the other server
- Primary Server stores a file about its zone. It has authority to create, maintain, and update the zone file.
- Secondary Server transfers complete information about a zone from another server which may be primary or secondary server. The secondary server does not have authority to create or update a zone file.
- DNS is a TCP/IP protocol used on different platforms. The domain name space is divided into three different sections: generic domains, country domains, and inverse domain.
- The main function of DNS is to translate domain names into IP Addresses, which computers can understand. It also provides a list of mail servers which accept Emails for each domain name. Each domain name in DNS will nominate a set of name servers to be authoritative for its DNS records.

ALGORITHM

Server

1. Start
2. Create UDP datagram socket
3. Create a table that maps host name and IP address
4. Receive the host name from the client
5. Retrieve the client's IP address from the received datagram
6. Get the IP address mapped for the host name from the table.
7. Display the host name and corresponding IP address

8. Send the IP address for the requested host name to the client
9. Stop.

Client

1. Start
2. Create UDP datagram socket.
3. Get the host name from the client
4. Send the host name to the server
5. Wait for the reply from the server
6. Receive the reply datagram and read the IP address for the requested host name
7. Display the IP address.
8. Stop.

PROGRAM

DNS Server

```
java import java.io.*;
import java.net.*;
public class udpdnsserver
{
	private static int indexOf(String[] array, String str)
	{
		str = str.trim();
		for (int i=0; i < array.length; i++)
		{
			if (array[i].equals(str))
				return i;
		}
		return -1;
	}
	public static void main(String arg[])throws IOException
	{
		String[] hosts = {"yahoo.com", "gmail.com","cricinfo.com", "facebook.com"};
		String[] ip = {"68.180.206.184", "209.85.148.19","80.168.92.140", "69.63.189.16"};
		System.out.println("Press Ctrl + C to Quit");
		while (true)
		{
			DatagramSocket serversocket=new DatagramSocket(1362);
			byte[] senddata = new byte[1021];
			byte[] receivedata = new byte[1021];
			DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);
			serversocket.receive(recvpack);
			String sen = new String(recvpack.getData());
```

```
				InetAddress ipaddress = recvpack.getAddress();
				int port = recvpack.getPort();
				String capsent;
				System.out.println("Request for host " + sen);
				if(indexOf (hosts, sen) != -1)
					capsent = ip[indexOf (hosts, sen)];
				else
					capsent = "Host Not Found";
				senddata = capsent.getBytes();
			DatagramPacket pack = new DatagramPacket (senddata, senddata.length,ipaddress,port);
				serversocket.send(pack);
				serversocket.close();
			}}}
```

UDP DNS Client

```
java import java.io.*;
import java.net.*;
public class udpdnsclient
{
	public static void main(String args[])throws IOException
	{
		BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
		DatagramSocket clientsocket = new DatagramSocket();
		InetAddress ipaddress;
		if (args.length == 0)
			ipaddress = InetAddress.getLocalHost();
		else
			ipaddress = InetAddress.getByName(args[0]);
			byte[] senddata = new byte[1024];
			byte[] receivedata = new byte[1024];
			int portaddr = 1362;
			System.out.print("Enter the hostname : ");
			String sentence = br.readLine();
			Senddata = sentence.getBytes();
			DatagramPacket pack = new DatagramPacket(senddata,senddata.length,
ipaddress,portaddr);
			clientsocket.send(pack);
			DatagramPacket recvpack =new DatagramPacket(receivedata,receivedata.length);
			clientsocket.receive(recvpack);
			String modified = new String(recvpack.getData());
			System.out.println("IP Address: " + modified);
			clientsocket.close();
```

}}

<u>OUTPUT</u>

<u>Server</u>

javac udpdnsserver.java
java udpdnsserver
Press Ctrl + C to Quit Request for host yahoo.com
Request for host cricinfo.com
Request for host youtube.com

Client

>javac udpdnsclient.java
>java udpdnsclient
Enter the hostname : yahoo.com
IP Address: 68.180.206.184
>java udpdnsclient
Enter the hostname : cricinfo.com
IP Address: 80.168.92.140
>java udpdnsclient
Enter the hostname : youtube.com
IP Address: Host Not Found

RESULT:

Thus the java application program using UDP Sockets to implement DNS was developed and executed successfully

Ex. No : 5 Date :	**Write a code simulating ARP /RARP protocols**

AIM:

To write a java program for simulating ARP and RARP protocols using TCP.

PRE LAB DISCUSSION:

- Address Resolution Protocol (ARP) is a low-level network protocol for translating network layer addresses into link layer addresses.ARP lies between layers 2 and 3 of the OSI model, although ARP was not included in the OSI framework and allows computers to introduce each other across a network prior to communication.Because protocols are basic network communication units, address resolution is dependent on protocols such as ARP, which is the only reliable method of handling required tasks.
- The Address Resolution Protocol (ARP) is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given internet layer address,
- When configuring a new network computer, each system is assigned an Internet Protocol (IP) address for primary identification and communication. A computer also has a unique media access control (MAC) address identity. Manufacturers embed the MAC address in the local area network (LAN) card. The MAC address is also known as the computer's physical address.
- Address Resolution Protocol (ARP) is used to resolve an IPv4 address (32 bit Logical Address) to the physical address (48 bit MAC Address). Network Applications at the Application Layer use IPv4 Address to communicate with another device.
- Reverse Address Resolution Protocol (RARP) is a network protocol used to resolve a data link layer address to the corresponding network layer address. For example, RARP is used to resolve a Ethernet MAC address to an IP address.
- The client broadcasts a RARP packet with an ethernet broadcast address, and it's own physical address in the data portion. The server responds by telling the client it's IP address. Note there is no name sent. Also note there is no security.
- Media Access Control (MAC) addresses need to be individually configured on the servers by an administrator. RARP is limited to serving only IP addresses.Reverse ARP differs from the Inverse Address Resolution Protocol which is designed to obtain the IP address associated with a local Frame Relay data link connection identifier. InARP is not used in Ethernet.

ALGORITHM:

Client

1. Start the program
2. Create socket and establish connection with the server.
3. Get the IP address to be converted into MAC address from the user.
4. Send this IP address to server.
5. Receive the MAC address for the IP address from the server.
6. Display the received MAC address
7. Terminate the connection

Server

1. Start the program
2. Create the socket, bind the socket created with IP address and port number and make it a listening socket.
3. Accept the connection request when it is requested by the client.
4. Server maintains the table in which IP and corresponding MAC addresses are stored.
5. Receive the IP address sent by the client.
6. Retrieve the corresponding MAC address for the IP address and send it to the client.
7. Close the connection with the client and now the server becomes a listening server waiting for the connection request from other clients
8. Stop

PROGRAM

Client:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientarp
{
	public static void main(String args[])
	{
	try
	{
		BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
		Socket clsct=new Socket("127.0.0.1",139)
		DataInputStream din=new DataInputStream(clsct.getInputStream());
		DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
		System.out.println("Enter the Logical address(IP):");
		String  str1=in.readLine();
		dout.writeBytes(str1+'\n';
		String str=din.readLine();
```

```
            System.out.println("The Physical Address is: "+str);
            clsct.close();
        }
        catch (Exception e)
        {
            System.out.println(e);
        }}
}
```

Server:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverarp
{
    public static void main(String args[])
    {
    try{
        ServerSocket obj=new
        ServerSocket(139); Socket
        obj1=obj.accept();
        while(true)
        {
            DataInputStream din=new DataInputStream(obj1.getInputStream());
            DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());
            String str=din.readLine();
            String ip[]={"165.165.80.80","165.165.79.1"};
            String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
            for(int i=0;i<ip.length;i++)
            {
                if(str.equals(ip[i]))
                {
                    dout.writeBytes(mac[i]+'\n');
                    break;
                }
            }
            obj.close();
        }
    }
    catch(Exception e)
    {
```

```
                System.out.println(e);
        }}
}
```

Output:

```
E:\networks>java Serverarp
E:\networks>java Clientarp
Enter the Logical address(IP):
165.165.80.80
The Physical Address is: 6A:08:AA:C2
```

(b) Program for Reverse Address Resolution Protocol (RARP) using UDP

ALGORITHM:

Client

1. Start the program
2. Create datagram socket
3. Get the MAC address to be converted into IP address from the user.
4. Send this MAC address to server using UDP datagram.
5. Receive the datagram from the server and display the corresponding IP address.
6. Stop

Server

1. Start the program.
2. Server maintains the table in which IP and corresponding MAC addresses are stored.
3. Create the datagram socket
4. Receive the datagram sent by the client and read the MAC address sent.
5. Retrieve the IP address for the received MAC address from the table.
6. Display the corresponding IP address.
7. Stop

PROGRAM:

Client:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientrarp12
{
        public static void main(String args[])
        {
```

```
try
{
        DatagramSocket client=new DatagramSocket();

        InetAddress addr=InetAddress.getByName("127.0.0.1");

        byte[] sendbyte=new byte[1024];

        byte[] receivebyte=new byte[1024];

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

        System.out.println("Enter the Physical address (MAC):")

        String str=in.readLine(); sendbyte=str.getBytes();
}
    DatagramPacket sender=newDatagramPacket(sendbyte,sendbyte.length,addr,1309);

    client.send(sender);

        DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);

        client.receive(receiver);

        String s=new String(receiver.getData());

        System.out.println("The Logical Address is(IP): "+s.trim());

        client.close();
```

```
catch(Exception e)
{
    System.out.println(e);
}}}
```

Server:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverrarp12
{
public static void main(String args[])
{
try{
    DatagramSocket server=new DatagramSocket(1309);
    while(true)
    {
        byte[] sendbyte=new byte[1024];
        byte[] receivebyte=new byte[1024];
        DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);
        server.receive(receiver);
        String str=new String(receiver.getData());
        String s=str.trim();
        InetAddress addr=receiver.getAddress();
        int port=receiver.getPort();

        String ip[]={"165.165.80.80","165.165.79.1"};
        String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
        for(int i=0;i<ip.length;i++)
        {
            if(s.equals(mac[i]))
            {
                sendbyte=ip[i].getBytes();
                DatagramPacket sender = new
                        DatagramPacket(sendbyte,sendbyte.length,addr,port);
                server.send(sender);
            }}  break;
        break;
```

```
}}}catch(Exception e)
{
        System.out.println(e);
}}}
```

Output:

I:\ex>java Serverrarp12
I:\ex>java Clientrarp12
Enter the Physical address (MAC):
6A:08:AA:C2
The Logical Address is(IP): 165.165.80.80

RESULT :

Thus the program for implementing to display simulating ARP and RARP protocols was executed successfully and output is verified.

Ex. No : 6 Date :	**Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS**

AIM:

To Study Network simulator (NS).and Simulation of Congestion Control Algorithms using NS

PRE LAB DISCUSSION:

NET WORK SIMULATOR (NS2)

Ns Overview

- Ns Status
- Periodical release (ns-2.26, Feb 2003)
- Platform support
- FreeBSD, Linux, Solaris, Windows and Mac

Ns functionalities

Routing, Transportation, Traffic sources,Queuing disciplines, QoS

Congestion Control Algorithms

- Slow start
- Additive increase/multiplicative decrease
- Fast retransmit and Fast recovery

Case Study: A simple Wireless network.

Ad hoc routing, mobile IP, sensor-MAC

Tracing, visualization and various utilitie

NS(Network Simulators)

Most of the commercial simulators are GUI driven, while some network simulators are CLI driven. The network model / configuration describes the state of the network (nodes,routers, switches, links) and the events (data transmissions, packet error etc.). An important output of simulations are the trace files. Trace files log every packet, every event that occurred in the simulation and are used for analysis. Network simulators can also provide other tools to facilitate visual analysis of trends and potential trouble spots.

Most network simulators use discrete event simulation, in which a list of pending "events" is stored, and those events are processed in order, with some events triggering future events—such as the event of the arrival of a packet at one node triggering the event of the arrival of that packet at a downstream node.

Simulation of networks is a very complex task. For example, if congestion is high, then estimation of the average occupancy is challenging because of high variance. To estimate the likelihood of a buffer overflow in a network, the time required for an accurate answer can be extremely large. Specialized techniques such as "control variates" and "importance sampling" have been developed to speed simulation.

Examples of network simulators

There are many both free/open-source and proprietary network simulators. Examples of notable network simulation software are, ordered after how often they are mentioned in research papers:

1. ns (open source)
2. OPNET (proprietary software)
3. NetSim (proprietary software)

Uses of network simulators

Network simulators serve a variety of needs. Compared to the cost and time involved in setting up an entire test bed containing multiple networked computers, routers and data links, network simulators are relatively fast and inexpensive. They allow engineers, researchers to test scenarios that might be particularly difficult or expensive to emulate using real hardware - for instance, simulating a scenario with several nodes or experimenting with a new protocol in the network. Network simulators are particularly useful in allowing researchers to test new networking protocols or changes to existing protocols in a controlled and reproducible environment. A typical network simulator encompasses a wide range of networking technologies and can help the users to build complex networks from basic building blocks such as a variety of nodes and links. With the help of simulators, one can design hierarchical networks using various types of nodes like computers, hubs, bridges, routers, switches, links, mobile units etc.

Various types of Wide Area Network (WAN) technologies like TCP, ATM, IP etc. and Local Area Network (LAN) technologies like Ethernet, token rings etc., can all be simulated with a typical simulator and the user can test, analyze various standard results apart from devising some novel protocol or strategy for routing etc. Network simulators are also widely used to simulate battlefield networks in Network-centric warfare.

There are a wide variety of network simulators, ranging from the very simple to the very complex. Minimally, a network simulator must enable a user to represent a network topology, specifying the nodes on the network, the links between those nodes and the traffic between the nodes. More complicated systems may allow the user to specify everything about the protocols used to handle traffic in a network. Graphical applications allow users to easily visualize the workings of their simulated environment. Text-based applications may provide a less intuitive interface, but may permit more advanced forms of customization.

Packet loss

Packet loss occurs when one or more packets of data travelling across a computer network fail to reach their destination. Packet loss is distinguished as one of the three main error types encountered in digital communications; the other two being bit error and spurious packets caused due to noise.

Packets can be lost in a network because they may be dropped when a queue in the network node overflows. The amount of packet loss during the steady state is another important property of a congestion control scheme. The larger the value of packet loss, the more difficult it is for transportlayer protocols to maintain high bandwidths, the sensitivity to loss of individual packets, as well as to frequency and patterns of loss among longer packet sequences is strongly dependent on the application itself.

Throughput

Throughput is the main performance measure characteristic, and most widely used. In communication networks, such as Ethernet or packet radio, throughput or network throughput is the average rate of successful message delivery over a communication channel. Throughput is usually measured inbitsper second (bit/s orbps), and sometimes in data packets per second or data packets per time slot. This measures how soon the receiver is able to get a certain amount of data send by the sender. It is determined as the ratio of the total data received to the end to end delay. Throughput is an important factor which directly impacts the network performance.

Delay

Delay is the time elapsed while a packet travels from one point e.g., source premise or network ingress to destination premise or network degrees. The larger the value of delay, the more difficult it is for transport layer protocols to maintain highbandwidths. We will calculate end to end delay

Queue Length

A queuing system in networks can be described as packets arriving for service, waiting for service if it is not immediate, and if having waited for service, leaving the system after being served. Thus queue length is very important characteristic to determine that how well the active queue management of the congestion control algorithm has been working.

Congestion control Algorithms

Slow-start is used in conjunction with other algorithms to avoid sending more data than the network is capable of transmitting, that is, to avoid causing network congestion. The additive increase/multiplicative decrease (AIMD) algorithm is a feedback control algorithm. AIMD combines linear growth of the congestion window with an exponential reduction when a congestion takes place. Multiple flows using AIMD congestion control will eventually converge to use equal amounts of a contended link. Fast Retransmit is an enhancement to TCP that reduces the time a sender waits before retransmitting a lost segment.

<u>Program:</u>

```
include <wifi_lte/wifi_lte_rtable.h>
  struct r_hist_entry *elm, *elm2;
int num_later = 1;
elm = STAILQ_FIRST(&r_hist_);
while (elm != NULL && num_later <= num_dup_acks_){
            num_later;
            elm = STAILQ_NEXT(elm, linfo_);
```

```
}

    if (elm != NULL){
        elm = findDataPacketInRecvHistory(STAILQ_NEXT(elm,linfo_));

        if (elm != NULL){
            elm2 = STAILQ_NEXT(elm, linfo_);
            while(elm2 != NULL){
                if (elm2->seq_num_ < seq_num && elm2->t_recv_ <
        time){

            STAILQ_REMOVE(&r_hist_,elm2,r_hist_entry,linfo_);
                        delete elm2;
                } else
                        elm = elm2;
                elm2 = STAILQ_NEXT(elm, linfo_);
            }
        }
}
}
void DCCPTFRCAgent::removeAcksRecvHistory(){
struct r_hist_entry *elm1 = STAILQ_FIRST(&r_hist_);
struct r_hist_entry *elm2;

int num_later = 1;
while (elm1 != NULL && num_later <= num_dup_acks_){
        num_later;
        elm1 = STAILQ_NEXT(elm1, linfo_);
}

if(elm1 == NULL)
        return;

elm2 = STAILQ_NEXT(elm1, linfo_);
while(elm2 != NULL){
        if (elm2->type_ == DCCP_ACK){
            STAILQ_REMOVE(&r_hist_,elm2,r_hist_entry,linfo_);
            delete elm2;
        } else {
            elm1 = elm2;
        }
```

```
        elm2 = STAILQ_NEXT(elm1, linfo_);
}
}
inline r_hist_entry
        *DCCPTFRCAgent::findDataPacketInRecvHistory(r_hist_entry *start){
while(start != NULL && start->type_ == DCCP_ACK)
        start = STAILQ_NEXT(start,linfo_);
return start;
}
```

Result:

Thus we have Studied Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.

Ex. No : 7 Date :	**Study of TCP/UDP performance using Simulation tool.**

AIM:

To simulate the performance of TCP/UDP using NS2.

PRE LAB DISCUSSION:

- TCP is reliable protocol. That is, the receiver always sends either positive or negative acknowledgement about the data packet to the sender, so that the sender always has bright clue about whether the data packet is reached the destination or it needs to resend it.
- TCP ensures that the data reaches intended destination in the same order it was sent.
- TCP is connection oriented. TCP requires that connection between two remote points be established before sending actual data.
- TCP provides error-checking and recovery mechanism.
- TCP provides end-to-end communication.
- TCP provides flow control and quality of service.
- TCP operates in Client/Server point-to-point mode.
- TCP provides full duplex server, i.e. it can perform roles of both receiver and sender.
- The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.UDP is used when acknowledgement of data does not hold any significance.
- UDP is good protocol for data flowing in one direction.
- UDP is simple and suitable for query based communications.
- UDP is not connection oriented.
- UDP does not provide congestion control mechanism.
- UDP does not guarantee ordered delivery of data.
- UDP is stateless.
- UDP is suitable protocol for streaming applications such as VoIP, multimedia streaming.

TCP Performance

Algorithm

1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.

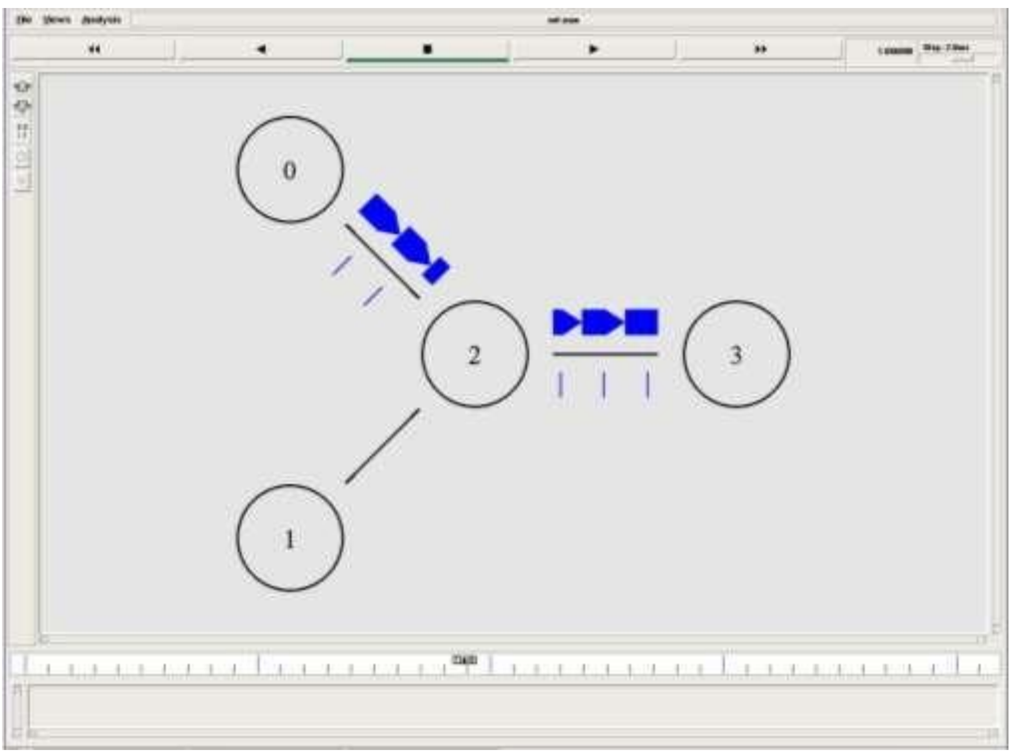
7. Create the applications and attach them to the tcp agent.Connect tcp and tcp sink.
8. Run the simulation.

PROGRAM:

```
set ns [new Simulator]
$ns color 0 Blue
$ns color 1 Red
$ns color 2
Yellowset n0
[$ns node] set
n1 [$ns node]
set n2 [$ns
node] set n3
[$ns node]
set f [open tcpout.tr w]
$ns trace-all $f
set nf [open tcpout.nam w]
$ns namtrace-all $nf
$ns duplex-link $n0 $n2 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n2 $n3
queuePos 0.5set tcp [new
Agent/TCP]
$tcp set class_ 1
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $tcp
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
```

```
$ns at 1.2 "$ftp start"
$ns at 1.35 "$ns detach-agent $n1 $tcp ; $ns detach-agent $n3 $sink"
$ns at 3.0
"finish"proc
finish {} {
        global ns f nf
        $ns flush-traceclose $f
        close $nf
        puts "Running nam.."
        exec xgraph tcpout.tr -geometry
        600x800 &exec nam tcpout.nam &
        exit 0
}
$ns run
```

Output



UDP Performance

ALGORITHM :

1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.

7. Create the applications and attach them to the UDP agent.
8. Connect udp and null agents.
9. Run the simulation.

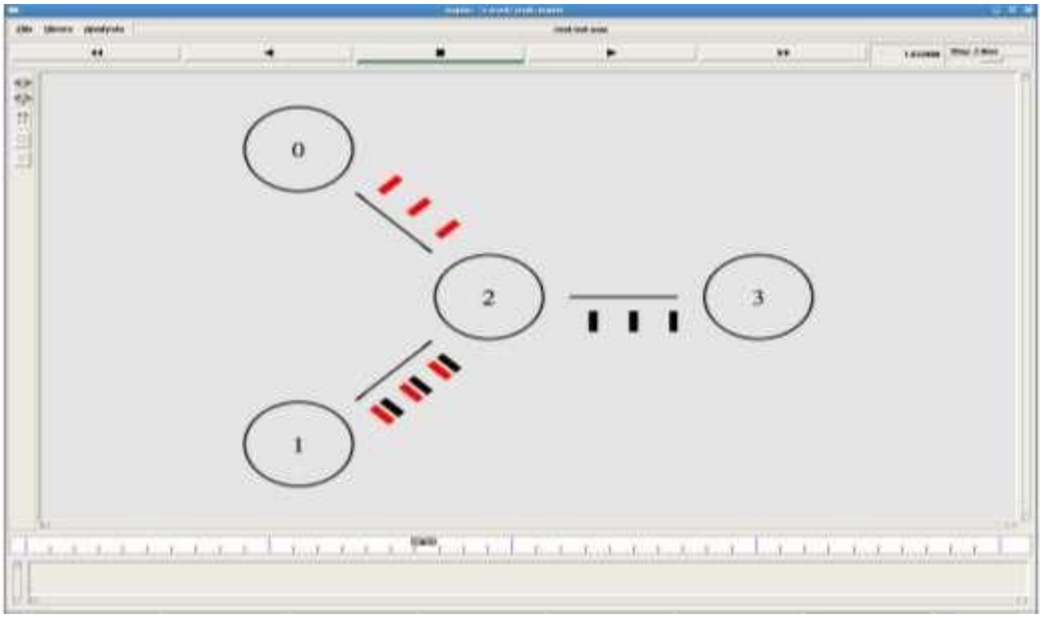
PROGRAM:

```
set ns [new Simulator]
$ns color 0 Blue
$ns color 1 Red
$ns color 2
Yellowset n0
[$ns node] set
n1 [$ns node]
set n2 [$ns
node] set n3
[$ns node]
set f [open udpout.tr w]
$ns trace-all $f
set nf [open udpout.nam w]
$ns namtrace-all $nf
$ns duplex-link $n0 $n2 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n2 $n3
queuePos 0.5set udp0 [new
Agent/UDP]
```

```
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent
$udp0 set udp1 [new
Agent/UDP]
$ns attach-agent $n3 $udp1

$udp1 set class_ 0


set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent
$udp1 set null0 [new
Agent/Null]
$ns attach-agent $n1
$null0set null1 [new
Agent/Null]
$ns attach-agent $n1 $null1
$ns connect $udp0 $null0
$ns connect $udp1 $null1
$ns at 1.0 "$cbr0 start"
$ns at 1.1 "$cbr1 start"
puts [$cbr0 set
packetSize_]puts [$cbr0
set interval_]
$ns at 3.0
"finish"proc
finish {} {
        global ns f nf
        $ns flush-
        traceclose $f
        close $nf
        puts "Running
        nam.." exec nam
        udpout.nam &exit 0
}
```

$ns run

Output:



RESULT :

Thus the study of TCP/UDP performance is done successfully.

Ex. No : 8 Date :	**Simulation of Distance Vector/ Link State Routing algorithm.**

AIM:

To simulate the Distance vector and link state routing protocols using NS2.

PRE LAB DISCUSSION:

LINK STATE ROUTING

Routing is the process of selecting best paths in a network. In the past, the term routing was also used to mean forwarding network traffic among networks. However this latter function is much better described as simply forwarding. Routing is performed for many kinds of networks, including the telephone network (circuit switching), electronic data networks (such as the Internet), and transportation networks. This article is concerned primarily with routing in electronic data networks using packet switching technology.

In packet switching networks, routing directs packet forwarding (the transit of logically addressed network packets from their source toward their ultimate destination) through intermediate nodes. Intermediate nodes are typically network hardware devices such as routers, bridges, gateways, firewalls, or switches. General-purpose computers can also forward packets and perform routing, though they are not specialized hardware and may suffer from limited performance. The routing process usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the router's memory, is very important for efficient routing. Most routing algorithms use only one network path at a time. Multipath routing techniques enable the use of multiple alternative paths.

In case of overlapping/equal routes, the following elements are considered in order to decide which routes get installed into the routing table (sorted by priority):

1. *Prefix-Length*: where longer subnet masks are preferred (independent of whether it is within a routing protocol or over different routing protocol)
2. *Metric*: where a lower metric/cost is preferred (only valid within one and the same routing protocol)
3. *Administrative distance*: where a lower distance is preferred (only valid between different routing protocols)

Routing, in a more narrow sense of the term, is often contrasted with bridging in its assumption that network addresses are structured and that similar addresses imply proximity within the network. Structured addresses allow a single routing table entry to represent the route to a group of devices. In large networks, structured addressing (routing, in the narrow sense) outperforms unstructured addressing (bridging). Routing has become the dominant form of addressing on the Internet. Bridging is still widely used within localized environments.

b. Flooding

Flooding is a simple routing algorithm in which every incoming packet is sent through every outgoing link except the one it arrived on. Flooding is used in bridging and in systems such as Usenet and peer-to-peer file sharing and as part of some routing protocols, including OSPF, DVMRP, and those used in ad-hoc wireless networks.There are generally two types of flooding available, Uncontrolled Flooding and Controlled Flooding. Uncontrolled Flooding is the fatal law of flooding. All nodes have neighbours and route packets indefinitely. More than two neighbours creates a broadcast storm.

Controlled Flooding has its own two algorithms to make it reliable, SNCF (Sequence Number Controlled Flooding) and RPF (Reverse Path Flooding). In SNCF, the node attaches its own address and sequence number to the packet, since every node has a memory of addresses and sequence numbers. If it receives a packet in memory, it drops it immediately while in RPF, the node will only send the packet forward. If it is received from the next node, it sends it back to the sender.

Distance vector Routing:

In computer communication theory relating to packet-switched networks, a **distance-vector routing protocol** is one of the two major classes of routing protocols, the other major class being the link-state protocol. Distance-vector routing protocols use the Bellman–Ford algorithm, Ford–Fulkerson algorithm, or DUAL FSM (in the case of Cisco Systems's protocols) to calculate paths.

A distance-vector routing protocol requires that a router informs its neighbors of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.

The term *distance vector* refers to the fact that the protocol manipulates *vectors* (arrays) of distances to other nodes in the network. The vector distance algorithm was the original ARPANET routing algorithm and was also used in the internet under the name of RIP (Routing Information Protocol).Examples of distance-vector routing protocols include RIPv1 and RIPv2 and IGRP.

Method

Routers using distance-vector protocol do not have knowledge of the entire path to a destination. Instead they use two methods:

1. Direction in which router or exit interface a packet should be forwarded.
2. Distance from its destination

Distance-vector protocols are based on calculating the direction and distance to any link in a network. "Direction" usually means the next hop address and the exit interface. "Distance" is a measure of the cost to reach a certain node. The least cost route between any two nodes is the route with minimum distance. Each node maintains a vector (table) of minimum distance to every node. The cost of reaching a destination is calculated using various route metrics. RIP uses the hop count of the destination whereas IGRP takes into account other information such as node delay and available bandwidth.

Updates are performed periodically in a distance-vector protocol where all or part of a router's routing table is sent to all its neighbors that are configured to use the same distance-vector routing protocol. RIP supports cross-platform distance vector routing whereas IGRP is a Cisco Systems proprietary distance vector routing protocol. Once a router has this information it is able to amend its own routing table to reflect the changes and then inform its neighbors of the changes. This process has been described as ‗routing by rumor' because routers are relying on the information they receive from other routers and cannot determine if the information is actually valid and true. There are a number of features which can be used to help with instability and inaccurate routing information.

EGP and BGP are not pure distance-vector routing protocols because a distance-vector protocol calculates routes based only on link costs whereas in BGP, for example, the local route preference value takes priority over the link cost.

Count-to-infinity problem

The Bellman–Ford algorithm does not prevent routing loops from happening and suffers from the **count-to-infinity problem**. The core of the count-to-infinity problem is that if A tells B that it has a path somewhere, there is no way for B to know if the path has B as a part of it. To see the problem clearly, imagine a subnet connected like A–B–C–D–E–F, and let the metric between the routers be "number of jumps". Now suppose that A is taken offline. In the vector-update-process B notices that the route to A, which was distance 1, is down – B does not receive the vector update from A. The problem is, B also gets an update from C, and C is still not aware of the fact that A is down – so it tells B that A is only two jumps from C (C to B to A), which is false. This slowly propagates through the network until it reaches infinity (in which case the algorithm corrects itself, due to the relaxation property of Bellman–Ford).

ALGORITHM:

1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.
7. Create the applications and attach them to the udp agent.
8. Connect udp and null..
9. At 1 sec the link between node 1 and 2 is broken.
10. At 2 sec the link is up again.
11. Run the simulation.

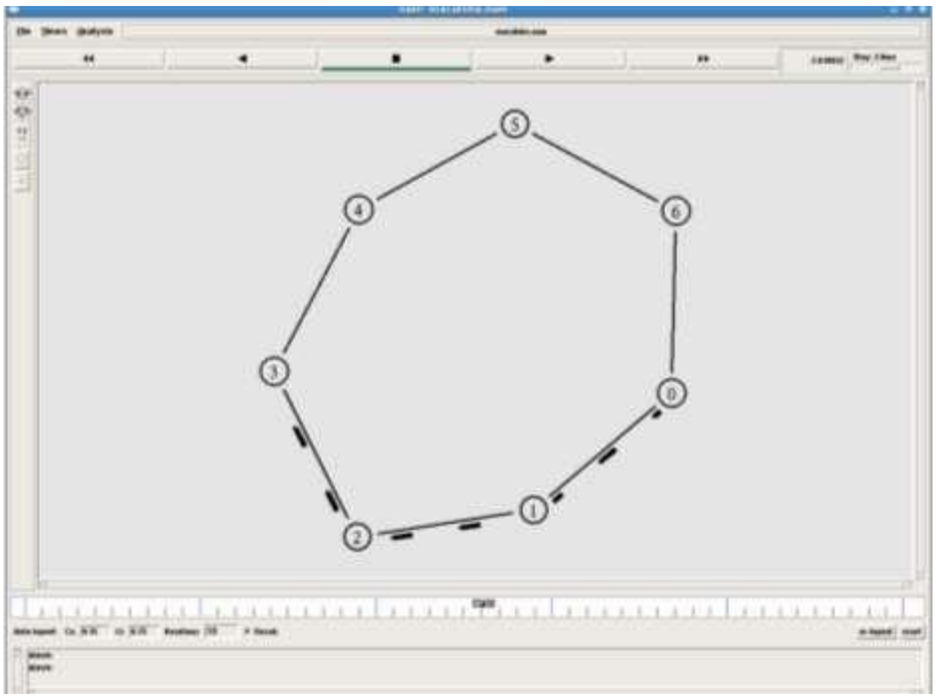
LINK STATE ROUTING PROTOCOL

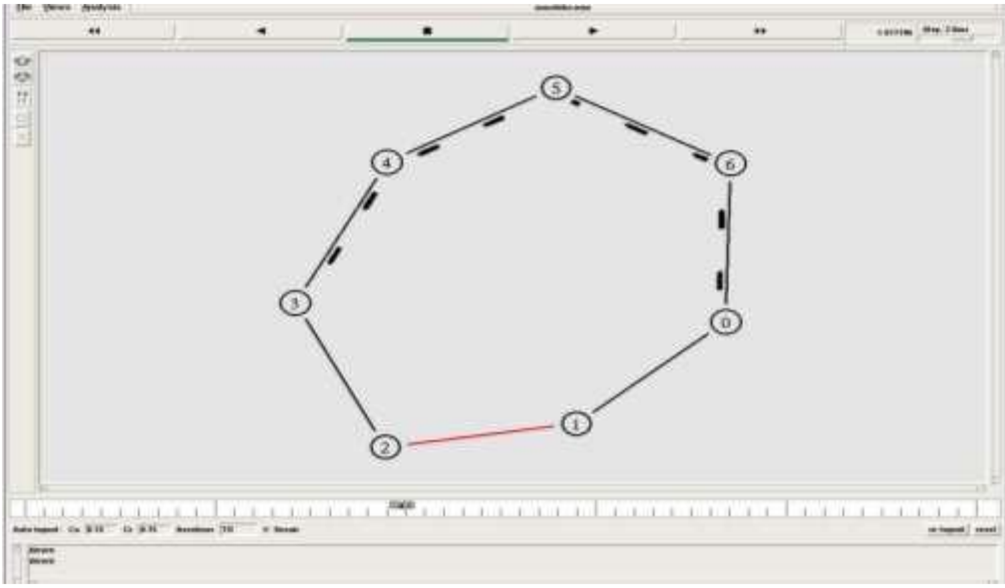
PROGRAM

```
set ns [new Simulator]
$ns rtproto LS
set nf [open linkstate.nam w]
$ns namtrace-all $nf
set f0 [open linkstate.tr w]
$ns trace-all
$f0proc finish
{} {
        global ns f0 nf
        $ns flush-
        traceclose
        $f0
        close $nf
        exec nam
        linkstate.nam &exit 0
}
for {set i 0} {$i <7} {incr
        i} {set n($i) [$ns
        node]
}
for {set i 0} {$i <7} {incr i} {
        $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent
$udp0 set null0 [new
Agent/Null]
$ns attach-agent $n(3) $null0
$ns connect $udp0 $null0
```

```
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"$ns at 5.0 "finish"
$ns run
```

Output:





DISTANCE VECTOR ROUTING ALGORITHM

ALGORITHM:

1. Create a simulator object
2. Set routing protocol to Distance Vector routing
3. Trace packets on all links onto NAM trace and text trace file
4. Define finish procedure to close files, flush tracing and run NAM
5. Create eight nodes
6. Specify the link characteristics between nodes
7. Describe their layout topology as a octagon
8. Add UDP agent for node n1
9. Create CBR traffic on top of UDP and set traffic parameters.
10. Add a sink agent to node n4
11. Connect source and the sink
12. Schedule events as follows:
    a. Start traffic flow at 0.5
    b. Down the link n3-n4 at 1.0
    c. Up the link n3-n4 at 2.0
    d. Stop traffic at 3.0
    e. Call finish procedure at 5.0
13. Start the scheduler
14. Observe the traffic route when link is up and down
15. View the simulated events and trace file analyze it
16. Stop

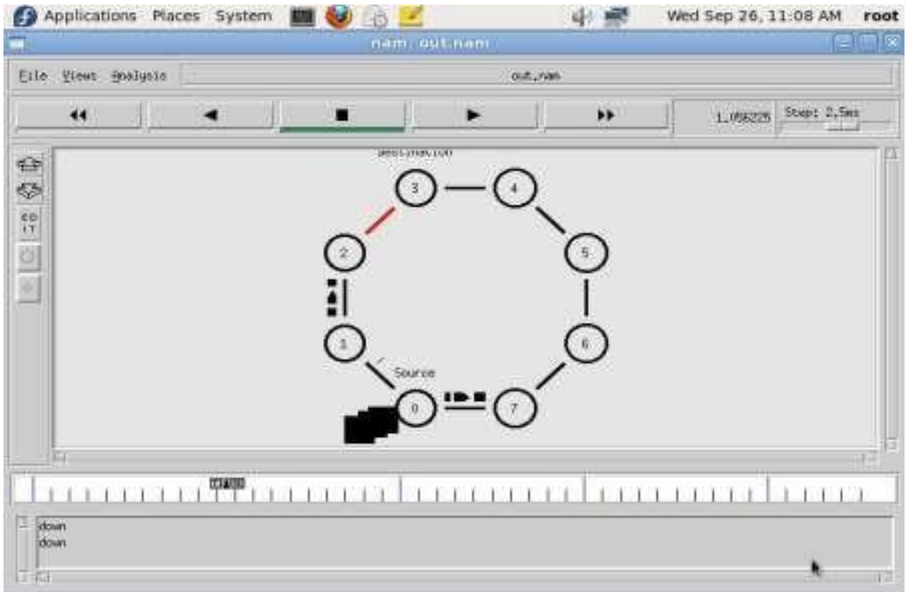
PROGRAM

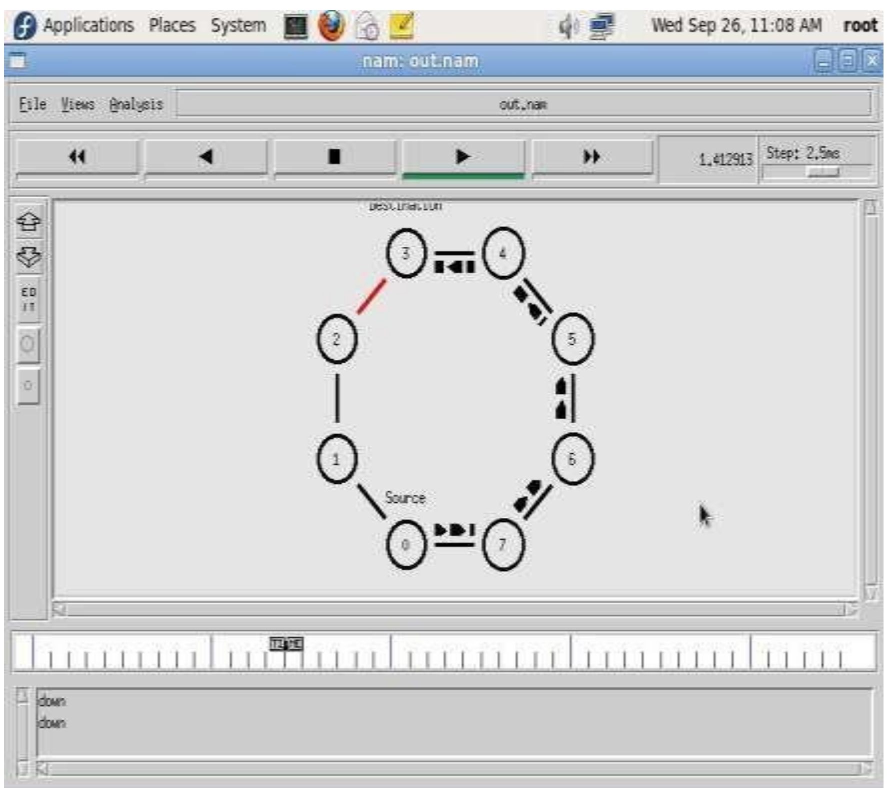
```
#Distance vector routing protocol – distvect.tcl
#Create a simulator object
set ns [new Simulator]
#Use distance vector routing
$ns rtproto DV
#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf
# Open tracefile
set nt [open trace.tr w]
$ns trace-all $nt
#Define 'finish' procedure
proc finish {}
{
	global ns nf
	$ns flush-trace
	#Close the trace file
	close $nf
	#Execute nam on the trace file
	exec nam -a out.nam &
	exit 0
}
# Create 8 nodes
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
# Specify link characterestics
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns duplex-link $n5 $n6 1Mb 10ms DropTail
$ns duplex-link $n6 $n7 1Mb 10ms DropTail
$ns duplex-link $n7 $n8 1Mb 10ms DropTail
$ns duplex-link $n8 $n1 1Mb 10ms DropTail
# specify layout as a octagon
$ns duplex-link-op $n1 $n2 orient left-up
$ns duplex-link-op $n2 $n3 orient up
```

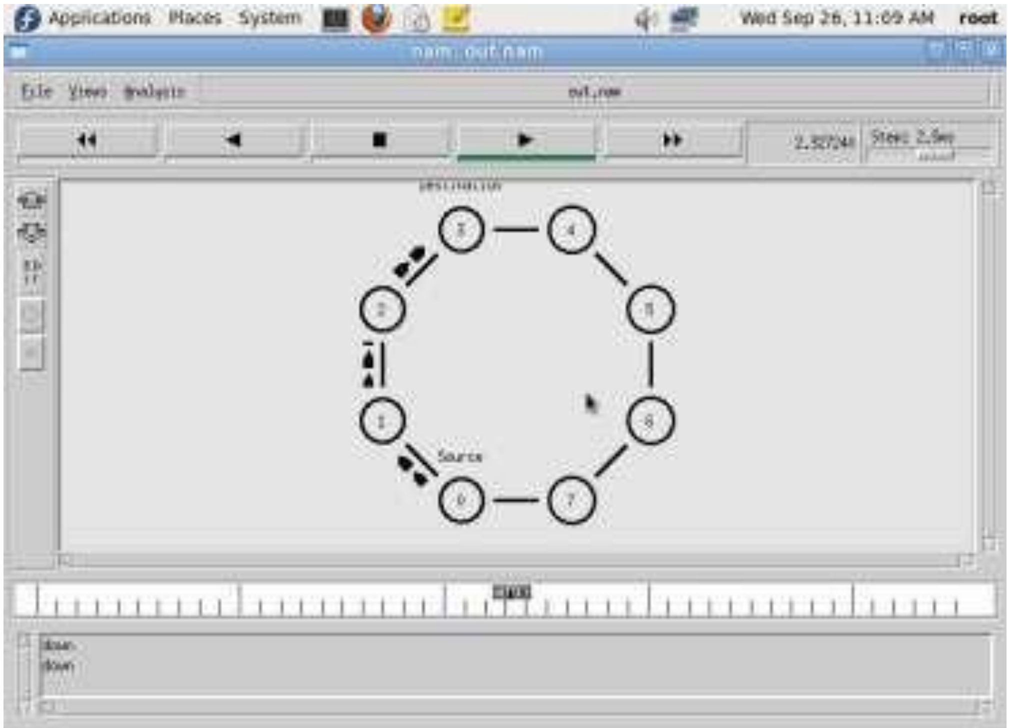
```
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n4 $n5 orient right
$ns duplex-link-op $n5 $n6 orient right-down
$ns duplex-link-op $n6 $n7 orient down
$ns duplex-link-op $n7 $n8 orient left-down
$ns duplex-link-op $n8 $n1 orient left
#Create a UDP agent and attach it to node n1
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
#Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
#Create a Null agent (a traffic sink) and attach it to node n4
set null0 [new Agent/Null]
$ns attach-agent $n4 $null0
#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0
#Schedule events for the CBR agent and the network dynamics
$ns at 0.0 "$n1 label Source"
$ns at 0.0 "$n4 label Destination"
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n3 $n4
$ns rtmodel-at 2.0 up $n3 $n4
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

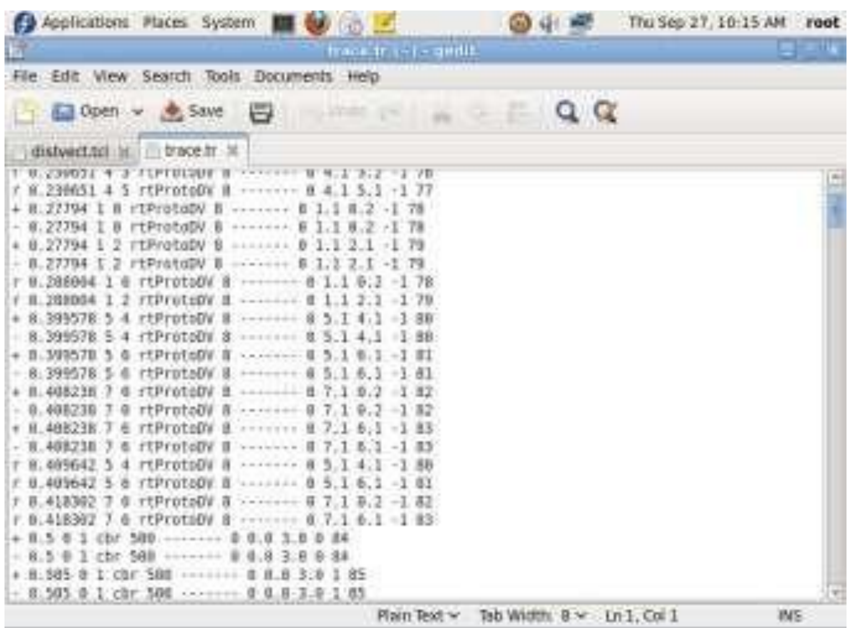
$ **ns distvect.tcl**

OUTPUT


Applications Places System
Wed Sep 26, 11:08 AM root
nam: out.nam
File Views Analysis
out.nam
Step: 2.5ms
Source
down
down


Applications Places System
Wed Sep 26, 11:08 AM root
nam: out.nam
File Views Analysis
out.nam
1.412913
Step: 2.5ms
Source
down
down





RESULT:

Thus the simulation for Distance vector and link state routing protocols was done using NS2.

Performance Evaluation of Routing protocols using Simulation tool.

Ex. No : 9(a) **Date :**	**UNICAST ROUTING PROTOCOL**

AIM:

To write a ns2 program for implementing unicast routing protocol.

PRE LAB DISCUSSION:

- When a device has multiple paths to reach a destination, it always selects one path by preferring it over others. This selection process is termed as Routing. Routing is done by special network devices called routers or it can be done by means of software processes.
- The software based routers have limited functionality and limited scope.A router is always configured with some default route. A default route tells the router where to forward a packet if there is no route found for specific destination.
- In case there are multiple path existing to reach the same destination, router can make decision based on the following information.Routes can be statically configured or dynamically learnt. One route can be configured to be preferred over others.Most of the traffic on the internet and intranets known as unicast data or unicast traffic is sent with specified destination. Routing unicast data over the internet is called unicast routing.
- It is the simplest form of routing because the destination is already known. Hence the router just has to look up the routing table and forward the packet to next hop.
- Multicasting in computer network is a group communication, where a sender(s) send data to multiple receivers simultaneously. It supports one – to – many and many – to – many data transmission across LANs or WANs. Through the process of multicasting, the communication and processing overhead of sending the same data packet or data frame in minimized.
- Multicast IP Routing protocols are used to distribute data (for example, audio/video streaming broadcasts) to multiple recipients. Using multicast, a source can send a single copy of data to a single multicast address, which is then distributed to an entire group of recipients.
- The key difference between broadcast and multicast is that in the broadcast the packet is delivered to all the host connected to the network whereas, in multicast packet is delivered to intended recipients only.
- Multicast Message. Multicasting identifies logical groups of computers. A single message can then be sent to the group. Multicast Message. Multicasting uses the Internet Group Management Protocol (IGMP) to identify groups and group members.

ALGORITHM:

1. Start the program.
2. Declare the global variables ns for creating a new simulator.
3. Set the color for packets.
4. Open the network animator file in the name of file2 in the write mode.
5. Open the trace file in the name of file 1 in the write mode.
6. Set the unicast routing protocol to transfer the packets in network.
7. Create the required no of nodes.
8. Create the duplex-link between the nodes including the delay time,bandwidth and dropping queue mechanism.
9. Give the position for the links between the nodes.
10. Set a tcp reno connection for source node.
11. Set the destination node using tcp sink.
12. Setup a ftp connection over the tcp connection.
13. Down the connection between any nodes at a particular time.
14. Reconnect the downed connection at a particular time.
15. Define the finish procedure.
16. In the definition of the finish procedure declare the global variables ns, file1, and file2.
17. Close the trace file and name file and execute the network animation file.
18. At the particular time call the finish procedure.
19. Stop the program.

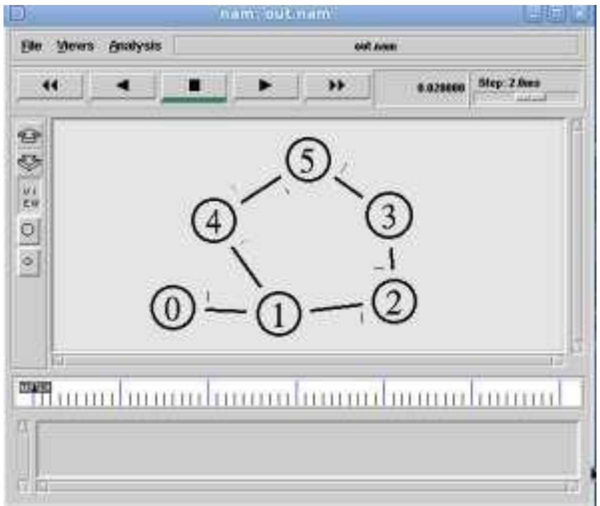
PROGRAM:

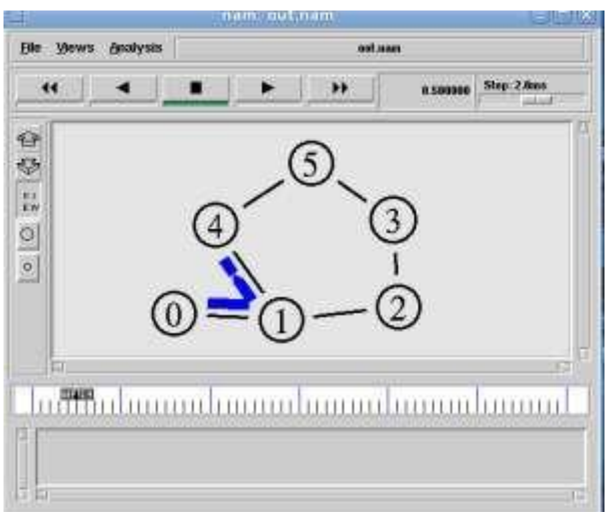
```
set ns [new Simulator]
#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
#Open the Trace file
set file1 [open out.tr w]
$ns trace-all $file1
#Open the NAM trace file
set file2 [open out.nam w]
$ns namtrace-all $file2
#Define a 'finish' procedure
proc finish {}
{
	global ns file1 file2
	$ns flush-trace
	close $file1
	close $file2
	exec nam out.nam &
```

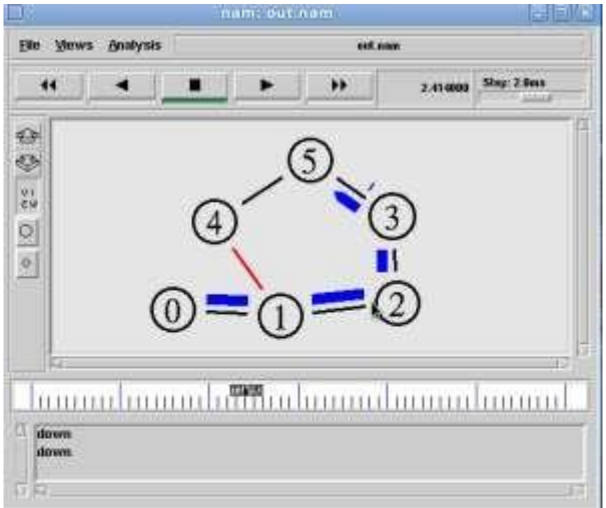
```
        exit 3
}
# Next line should be commented out to have the static routing
$ns rtproto DV
#Create six nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n4 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
#Create links between the nodes
$ns duplex-link $n0 $n1 0.3Mb 10ms DropTail
$ns duplex-link $n1 $n2 0.3Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.3Mb 10ms DropTail
$ns duplex-link $n1 $n4 0.3Mb 10ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 10ms DropTail
$ns duplex-link $n4 $n5 0.5Mb 10ms DropTail

#Give node position (for NAM)
$ns duplex-link-op  $n0 $n1 orient right
$ns duplex-link-op  $n1 $n2 orient right
$ns duplex-link-op $n2 $n3 orient up
$ns duplex-link-op $n1 $n4 orient up-left
$ns duplex-link-op  $n3 $n5 orient left-up
$ns duplex-link-op  $n4 $n5 orient right-up

#Setup a TCP connection
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

$ns rtmodel-at 1.0 down $n1 $n4
```

```
$ns rtmodel-at 4.5 up $n1 $n4
$ns at 0.1 "$ftp start"
$ns at 6.0 "finish"
$ns run
```

Output :







RESULT:

Thus the case study about the different unicast routing algorithms to select the network path withits optimum and economical during data transfer is done.

Ex. No : 9(b) Date :	**MULTICASTING ROUTING PROTOCOL**

AIM:

To write a ns2 program for implementing multicasting routing protocol.

ALGORITHM:

1. Start the program.
2. Declare the global variables ns for creating a new simulator.
3. Set the color for packets.
4. Open the network animator file in the name of file2 in the write mode.
5. Open the trace file in the name of file 1 in the write mode.
6. Set the multicast routing protocol to transfer the packets in network.
7. Create the multicast capable no of nodes.
8. Create the duplex-link between the nodes including the delay time,bandwidth and dropping queue mechanism.
9. Give the position for the links between the nodes.
10. Set a udp connection for source node.
11. Set the destination node ,port and random false for the source and destination files.
12. Setup a traffic generator CBR for the source and destination files.
13. Down the connection between any nodes at a particular time.
14. Create the receive agent for joining and leaving if the nodes in the group.
15. Define the finish procedure.
16. In the definition of the finish procedure declare the global variables.
17. Close the trace file and namefile and execute the network animation file.
18. At the particular time call the finish procedure.
19. Stop the program.

PROGRAM:

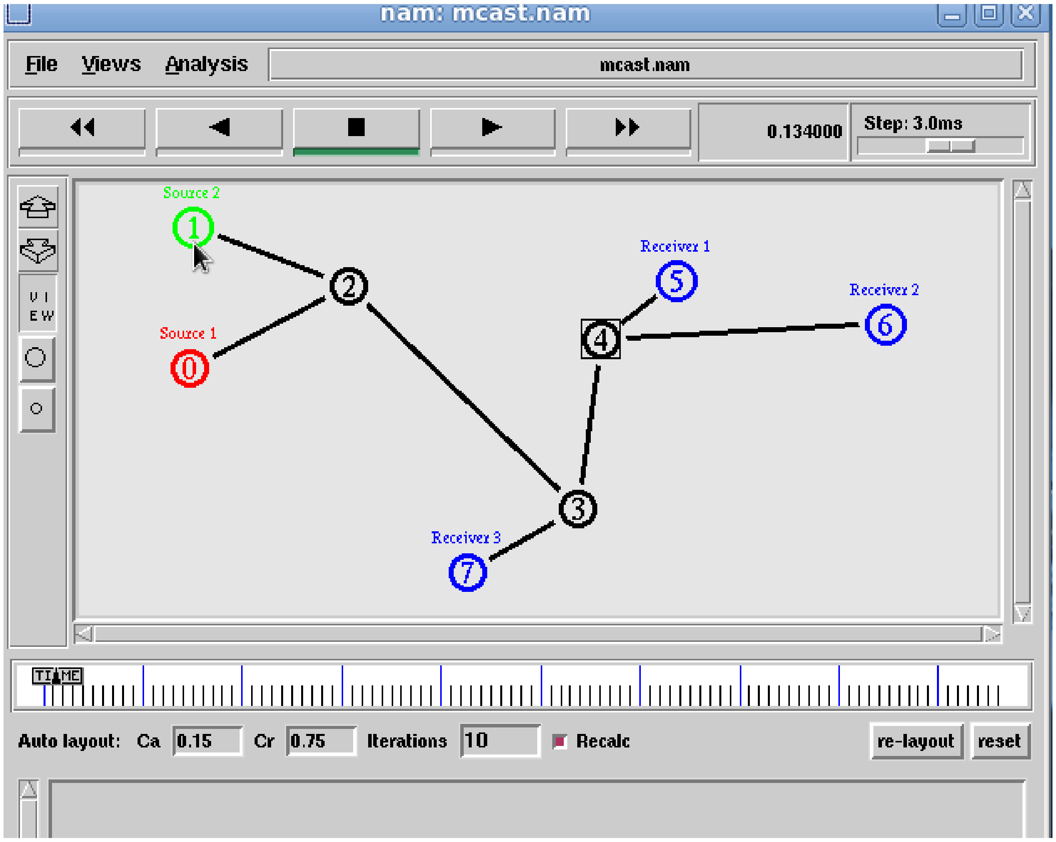
```
# Create scheduler
#Create an event scheduler wit multicast turned on
set ns [new Simulator -multicast on]
#$ns multicast
#Turn on Tracing
set tf [open output.tr w]
$ns trace-all $tf
# Turn on nam Tracing
set fd [open mcast.nam w]
$ns namtrace-all $fd
# Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
```

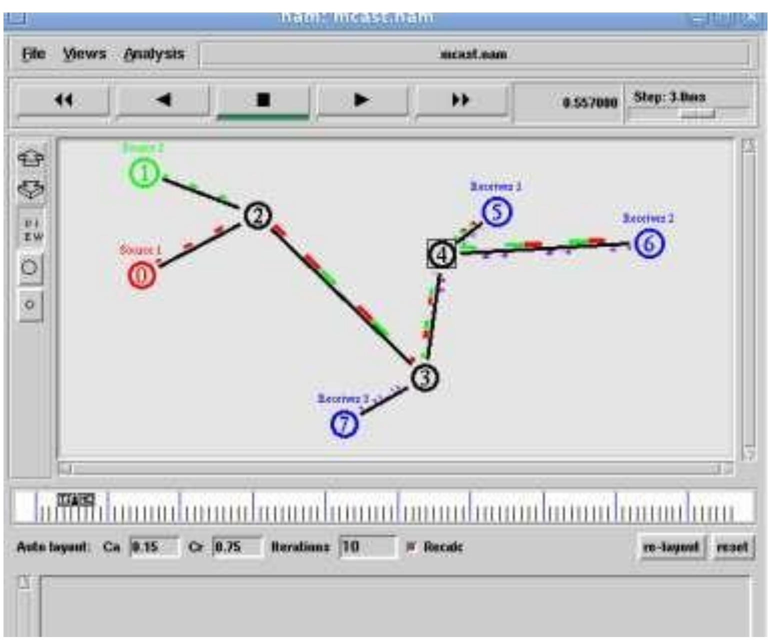
```
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]

# Create links
$ns duplex-link $n0 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n1 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
$ns duplex-link $n3 $n4 1.5Mb 10ms DropTail
$ns duplex-link $n3 $n7 1.5Mb 10ms DropTail
$ns duplex-link $n4 $n5 1.5Mb 10ms DropTail
$ns duplex-link $n4 $n6 1.5Mb 10ms DropTail

# Routing protocol: say distance vector
#Protocols: CtrMcast, DM, ST, BST
set mproto DM
set mrthandle [$ns mrtproto $mproto {}]

# Allocate group addresses
set group1 [Node allocaddr]
set group2 [Node allocaddr]

# UDP Transport agent for the traffic source
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
$udp0 set dst_addr_ $group1
$udp0 set dst_port_ 0
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp0

# Transport agent for the traffic source
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
$udp1 set dst_addr_ $group2
$udp1 set dst_port_ 0
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp1

# Create receiver
set rcvr1 [new Agent/Null]
$ns attach-agent $n5 $rcvr1
$ns at 1.0 "$n5 join-group $rcvr1 $group1"
set rcvr2 [new Agent/Null]
```

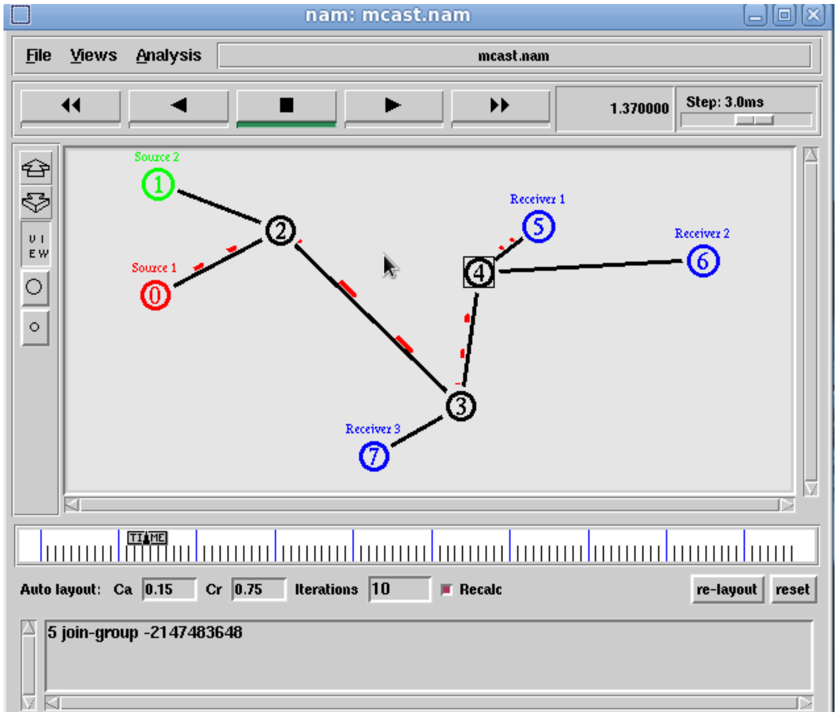
```
$ns attach-agent $n6 $rcvr2
$ns at 1.5 "$n6 join-group $rcvr2 $group1"
set rcvr3 [new Agent/Null]
$ns attach-agent $n7 $rcvr3
$ns at 2.0 "$n7 join-group $rcvr3 $group1"
set rcvr4 [new Agent/Null]
$ns attach-agent $n5 $rcvr1
$ns at 2.5 "$n5 join-group $rcvr4 $group2"
set rcvr5 [new Agent/Null]
$ns attach-agent $n6 $rcvr2
$ns at 3.0 "$n6 join-group $rcvr5 $group2"
set rcvr6 [new Agent/Null]
$ns attach-agent $n7 $rcvr3
$ns at 3.5 "$n7 join-group $rcvr6 $group2"
$ns at 4.0 "$n5 leave-group $rcvr1 $group1"
$ns at 4.5 "$n6 leave-group $rcvr2 $group1"
$ns at 5.0 "$n7 leave-group $rcvr3 $group1"
$ns at 5.5 "$n5 leave-group $rcvr4 $group2"
$ns at 6.0 "$n6 leave-group $rcvr5 $group2"
$ns at 6.5 "$n7 leave-group $rcvr6 $group2"

# Schedule events

$ns at 0.5 "$cbr1 start"
$ns at 9.5 "$cbr1 stop"
$ns at 0.5 "$cbr2 start"
$ns at 9.5 "$cbr2 stop"

#post-processing
$ns at 10.0 "finish"
proc finish {}
{

	global ns tf
	$ns flush-trace
	close $tf
	exec nam mcast.nam &
	exit 0
}

# For nam
#Colors for packets from two mcast groups
$ns color 10 red
$ns color 11 green
```

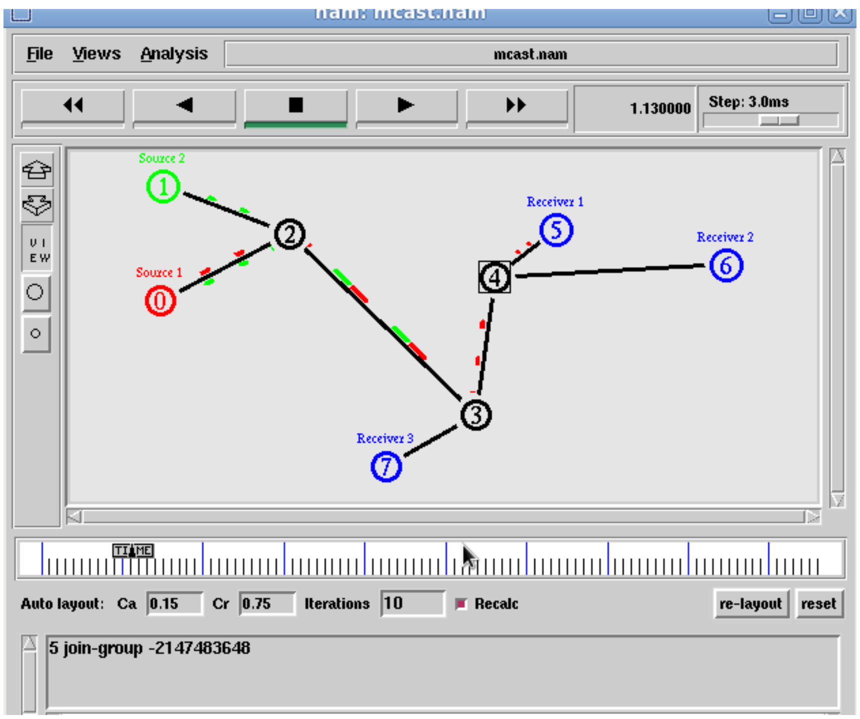
```
$ns color 30 purple
$ns color 31 green

# Manual layout: order of the link is significant!
#$ns duplex-link-op $n0 $n1 orient right
#$ns duplex-link-op $n0 $n2 orient right-up
#$ns duplex-link-op $n0 $n3 orient right-down
# Show queue on simplex link n0->n1
#$ns duplex-link-op $n2 $n3 queuePos 0.5

# Group 0 source
$udp0 set fid_ 10
$n0 color red
$n0 label "Source 1"

# Group 1 source
$udp1 set fid_ 11
$n1 color green
$n1 label "Source 2"
$n5 label "Receiver 1"
$n5 color blue
$n6 label "Receiver 2"
$n6 color blue
$n7 label "Receiver 3"
$n7 color blue

#$n2 add-mark m0 red
#$n2 delete-mark m0"

# Animation rate
$ns set-animation-rate 3.0ms
$ns run
```

Output:


nam: mcast.nam
File
Views
Analysis
mcast.nam
0.134000
Step: 3.0ms
Source 2
Source 1
Receiver 1
Receiver 2
Receiver 3
TIME
Auto layout:
Ca
0.15
Cr
0.75
Iterations
10
Recalc
re-layout
reset


File
Views
Analysis
mcast.nam
re-layout
reset





RESULT:

Thus the case study about the different multicasting routing algorithms to select the network path withits optimum and economical during data transfer is done.

Ex. No : 10 Date :	**Simulation of ErrorDetection Code (like CRC)**

AIM:

To implement error checking code using java.

PRE LAB DISSCUSSION:

The cyclic redundancy check, or CRC, is a technique for detecting errors in digital data, but not for making corrections when errors are detected. It is used primarily in data transmission.

In the CRC method, a certain number of check bits, often called a checksum, are appended to the message being transmitted. The receiver can determine whether or not the check bits agree with the data, to ascertain with a certain degree of probability whether or not an error occurred in transmission.

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.

CRC uses Generator Polynomial which is available on both sender and receiver side. An example generator polynomial is of the form like x3 + x + 1. This generator polynomial represents key 1011. Another example is x2 + 1 that represents key 101.

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

- The binary data is first augmented by adding k-1 zeros in the end of the data
- Use modulo-2 binary division to divide binary data by the key and store remainder ofdivision.
- Append the remainder at the end of the data to form the encoded data and send the same

Receiver Side (Check if there are errors introduced in transmission)

Perform modulo-2 division again and if remainder is 0, then there are no errors.

Modulo 2 Division:

- The process of modulo-2 binary division is the same as the familiar division process we usefor decimal numbers. Just that instead of subtraction, we use XOR here.
- In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).
- The result of the XOR operation (remainder) is (n-1) bits, which is used for the next stepafter 1 extra bit is pulled down to make it n bits long.
- When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which isappended at the sender side.

ALGORITHM:

1. Start the Program
2. Given a bit string, append 0S to the end of it (the number of 0s is the same as the degree of the generator polynomial) let B(x) be the polynomial corresponding to B.
3. Divide B(x) by some agreed on polynomial G(x) (generator polynomial) and determine the remainder R(x). This division is to be done using Modulo 2 Division.
4. Define T(x) = B(x) –R(x)
5. (T(x)/G(x) => remainder 0)
6. Transmit T, the bit string corresponding to T(x).
7. Let T' represent the bit stream the receiver gets and T'(x) the associated polynomial. The receiver divides T1(x) by G(x). If there is a 0 remainder, the receiver concludes T = T' and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission
8. Stop the Program

PROGRAM:

```
import java.io.*;
class crc_gen
{
public static void main(String args[]) throws IOException {
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int[] data;
int[] div;
int[] divisor;
int[] rem;
int[] crc;
int data_bits, divisor_bits, tot_length;
System.out.println("Enter number of data bits : "); data_bits=Integer.parseInt(br.readLine());
data=new int[data_bits];
System.out.println("Enter data bits : ");
for(int i=0; i<data_bits; i++)
data[i]=Integer.parseInt(br.readLine());
System.out.println("Enter number of bits in divisor : ");
divisor_bits=Integer.parseInt(br.readLine()); divisor=new int[divisor_bits];
System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
divisor[i]=Integer.parseInt(br.readLine());
System.out.print("Data bits are : ");
for(int i=0; i< data_bits; i++)
System.out.print(data[i]);
System.out.println();
```

```
System.out.print("divisor bits are : ");
for(int i=0; i< divisor_bits; i++)
System.out.print(divisor[i]);
System.out.println();
 */
tot_length=data_bits+divisor_bits-1;
div=new int[tot_length];
rem=new int[tot_length];
crc=new int[tot_length];
/*............................CRC GENERATION....................................*/
for(int i=0;i<data.length;i++)
 div[i]=data[i];
System.out.print("Dividend (after appending 0's) are : "); for(int i=0; i< div.length; i++)
System.out.print(div[i]);
System.out.println();
for(int j=0; j<div.length; j++){
rem[j] = div[j];
}
rem=divide(div, divisor, rem);
 for(int i=0;i<div.length;i++)
{

//append dividend and remainder

crc[i]=(div[i]^rem[i]);
}
 System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
System.out.print(crc[i]);

/*............................ERROR DETECTION................................*/
System.out.println();
System.out.println("Enter CRC code of "+tot_length+" bits : "); for(int i=0; i<crc.length; i++)
crc[i]=Integer.parseInt(br.readLine());
System.out.print("crc bits are : ");
for(int i=0; i< crc.length; i++)
System.out.print(crc[i]);
System.out.println();
for(int j=0; j<crc.length; j++){
rem[j] = crc[j];
}
rem=divide(crc, divisor, rem);
for(int i=0; i< rem.length; i++)
{
if(rem[i]!=0)
{

System.out.println("Error");
break;
```

```
}
if(i==rem.length-1)
System.out.println("No Error");
}
System.out.println("THANK YOU. ....)");
}
static int[] divide(int div[],int divisor[], int rem[])
{
int cur=0;
while(true)
{
for(int i=0;i<divisor.length;i++)
rem[cur+i]=(rem[cur+i]^divisor[i]);
while(rem[cur]==0 && cur!=rem.length-1)
cur++;
if((rem.length-cur)<divisor.length)
break;
}
return rem;
}
}
```

OUTPUT :
Enter number of data bits :
7
Enter data bits :
1
0
1
1
0
0
1
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Dividend (after appending 0's) are : 101100100
CRC code :
101100111
Enter CRC code of 9 bits :
1
0
1
1
0
0
1
0
1
crc bits are : 101100101
Error
THANK YOU…..)
BUILD SUCCESSFUL (total time: 1 minute 34 seconds)

RESULT:

Thus the above program for error checking code using was executed successfully.

TOPIC BEYOND SYLLABUS

Ex. No : 11(a) Date :	**Simulation of Go Back N protocol**

AIM:

To Simulate and to study of Go Back N protocol

PRE LAB DISCUSSION:

- Go Back N is a connection oriented transmission. The sender transmits the frames continuously. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size.
- The sender has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously.
- A station may send multiple frames as allowed by the window size.
- Receiver sends an ACK i if frame i has an error. After that, the receiver discards all incoming frames until the frame with error is correctly retransmitted.
- Go-Back-N Automatic Repeat reQuest (Go-Back-N ARQ), 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.
- Go – Back – N ARQ uses the concept of protocol pipelining, i.e. 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 2n−1. Consequently, the size of the sending window is 2n−1. Thus in order to accommodate a sending window size of 2n−1, an 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

ALGORITHM :

1. The source node transmits the frames continuously.
2. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size.
3. The source node has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously.
4. The size of the window depends on the protocol designer.
5. For the first frame, the receiving node forms a positive acknowledgement if the frame is

received without error.

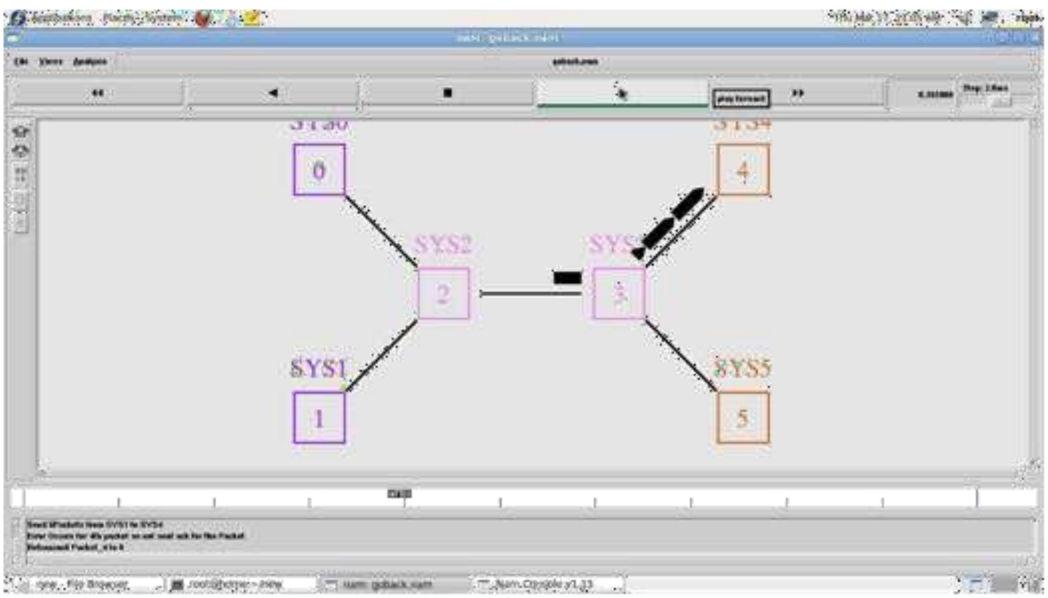
6. If subsequent frames are received without error (up to window size) cumulative positive acknowledgement is formed.
7. f the subsequent frame is received with error, the cumulative acknowledgment error-free frames are transmitted. If in the same window two frames or more frames are received with error, the second and the subsequent error frames are neglected. Similarly even the frames received without error after the receipt of a frame with error are neglected.
8. The source node retransmits all frames of window from the first error frame.
9. If the frames are errorless in the next transmission and if the acknowledgment is error free, the window slides by the number of error-free frames being transmitted.
10. If the acknowledgment is transmitted with error, all the frames of window at source are retransmitted, and window doesn't slide.
11. This concept of repeating the transmission from the first error frame in the window is called as GOBACKN transmission flow control protocol.

PROGRAM :

```
#send packets one by one
set ns [new Simulator] set n0 [$ns node]
set n1 [$ns node] set n2 [$ns node] set n3 [$ns node] set n4 [$ns node] set n5 [$ns node] $n0
color "purple" $n1 color "purple" $n2 color "violet" $n3 color "violet" $n4 color "chocolate" $n5
color "chocolate" $n0 shape box ;
$n1 shape box ; $n2 shape box ; $n3 shape box ; $n4 shape box ; $n5 shape box ;
$ns at 0.0 "$n0 label SYS0"
$ns at 0.0 "$n1 label SYS1"
$ns at 0.0 "$n2 label SYS2"
$ns at 0.0 "$n3 label SYS3"
$ns at 0.0 "$n4 label SYS4"
 $ns at 0.0 "$n5 label SYS5"
set nf [open goback.nam w] $ns namtrace-all $nf
set f [open goback.tr w] $ns trace-all $f
$ns duplex-link $n0 $n2 1Mb 20ms DropTail $ns duplex-link-op $n0 $n2 orient right-down $ns
queue-limit $n0 $n2 5
$ns duplex-link $n1 $n2 1Mb 20ms DropTail $ns duplex-link-op $n1 $n2 orient right-up $ns
duplex-link $n2 $n3 1Mb 20ms DropTail $ns duplex-link-op $n2 $n3 orient right
$ns duplex-link $n3 $n4 1Mb 20ms DropTail $ns duplex-link-op $n3 $n4 orient right-up $ns
duplex-link $n3 $n5 1Mb 20ms DropTail $ns duplex-link-op $n3 $n5 orient right-down
Agent/TCP set_nam_tracevar_true
set tcp [new Agent/TCP] $tcp set fid 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink] $ns attach-agent $n4 $sink $ns connect $tcp $sink
set ftp [new Application/FTP] $ftp attach-agent $tcp
$ns at 0.05 "$ftp start"
$ns at 0.06 "$tcp set windowlnit 6"
$ns at 0.06 "$tcp set maxcwnd 6"
  $ns at 0.25 "$ns queue-limit $n3 $n4 0"
$ns at 0.26 "$ns queue-limit $n3 $n4 10"
$ns at 0.305 "$tcp set windowlnit 4"

$ns at 0.305 "$tcp set maxcwnd 4"
```

```
$ns at 0.368 "$ns detach-agent $n1 $tcp ; $ns detach-agent $n4 $sink"
$ns at 1.5 "finish"
$ns at 0.0 "$ns trace-annotate \"Goback N end\""
$ns at 0.05 "$ns trace-annotate \"FTP starts at 0.01\""
$ns at 0.06 "$ns trace-annotate \"Send 6Packets from SYS1 to SYS4\""
$ns at 0.26 "$ns trace-annotate \"Error Occurs for 4th packet so not sent ack for the Packet\""
$ns at 0.30 "$ns trace-annotate \"Retransmit Packet_4 to 6\""
$ns at 1.0 "$ns trace-annotate \"FTP stops\""
proc finish {} {
global ns nf
$ns flush-trace
close $nf
puts "filtering..."
#exec tclsh../bin/namfilter.tcl goback.nam
#puts "running nam..."
exec nam goback.nam &
exit 0
}
$ns run.
```

OUTPUT



RESULT:

Thus the Go back N and Selective Repeat protocols were Simulated and studied

Ex. No : 11(b) Date :	**CARRIER SENSE MULTIPLE ACCESS**

AIM:

To write a ns2 program for implementing carrier sense multiple access.

PRE LAB DISCUSSION

Carrier Sensed Multiple Access (CSMA) : CSMA is a network access method used on shared network topologies such as Ethernet to control access to the network. Devices attached to the network cable listen (carrier sense) before transmitting. If the channel is in use, devices wait before transmitting. MA (Multiple Access) indicates that many devices can connect to and share the same network.

All devices have equal access to use the network when it is clear.In other words, a station that wants to communicate "listen" first on the media communication and awaits a "silence" of a preset time (called the Distributed Inter Frame Space or DIFS). After this compulsory period, the station starts a countdown for a random period considered. The maximum duration of this countdown is called the collision window (Window Collision, CW). If no equipment speaks before the end of the countdown, the station simply deliver its package.

However, if it is overtaken by another station, it stops immediately its countdown and waits for the next silence. then continued his account countdown where it left off. This is summarized in Figure. The waiting time random has the advantage of allowing a statistically equitable distribution of speaking time between the various network equipment, while making little unlikely (but not impossible) that both devices speak exactly the same time.

The countdown system prevents a station waiting too long before issuing its package. It's a bit what place in a meeting room when no master session (and all the World's polite) expected a silence, then a few moments before speaking, to allow time for someone else to speak. The time is and randomly assigned, that is to say, more or less equally.

ALGORITHM:

1. Start the program.
2. Declare the global variables ns for creating a new simulator.
3. Set the color for packets.
4. Open the network animator file in the write mode.
5. Open the trace file and the win file in the write mode.
6. Transfer the packets in network.
7. Create the capable no of nodes.
8. Create the duplex-link between the nodes including the delay time,bandwidth and dropping queue mechanism.
9. Give the position for the links between the nodes.
10. Set a tcp connection for source node.
11. Set the destination node using tcp sink.
12. Set the window size and the packet size for the tcp.

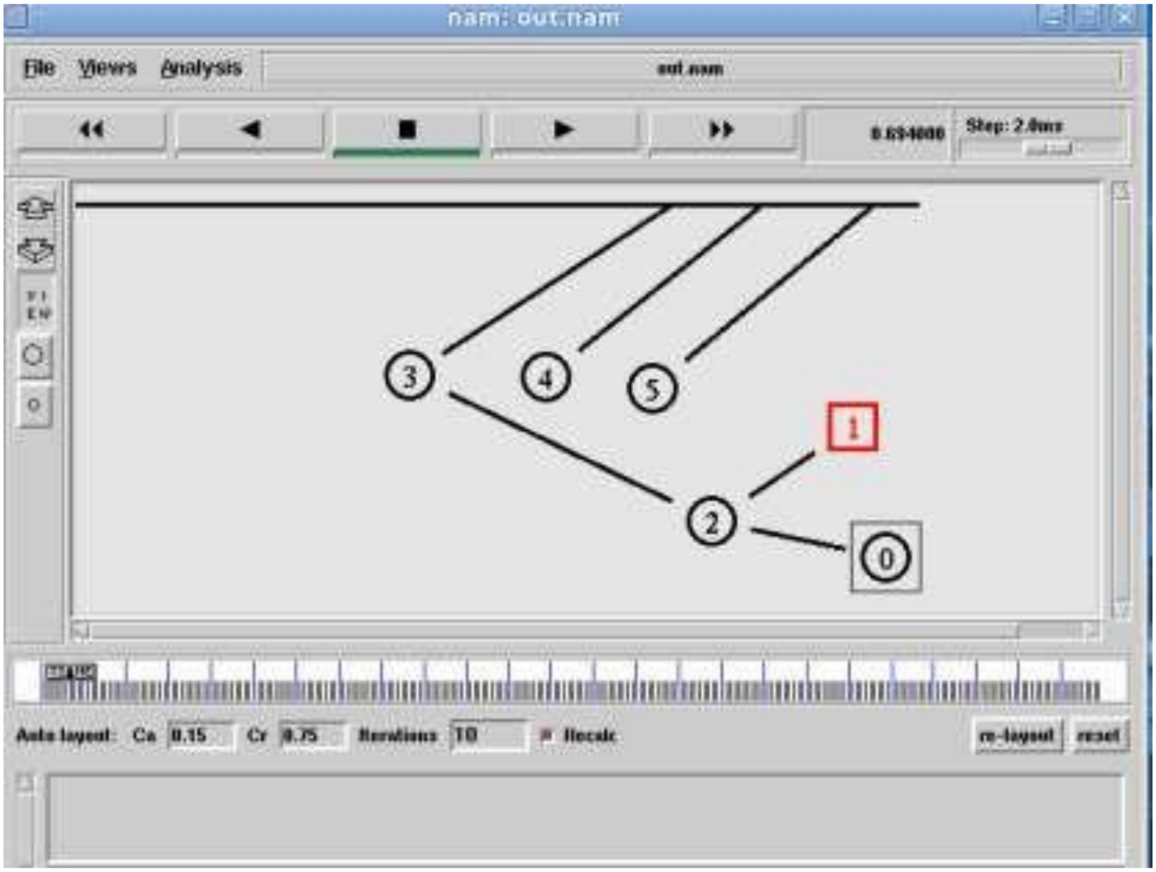
13. Set up the ftp over the tcp connection.
14. Set the udp and tcp connection for the source and destination.
15. Create the traffic generator CBR for the source and destination files.
16. Define the plot window and finish procedure.
17. In the definition of the finish procedure declare the global variables.
18. Close the trace file and namefile and execute the network animation file.
19. At the particular time call the finish procedure.
20. Stop the program.

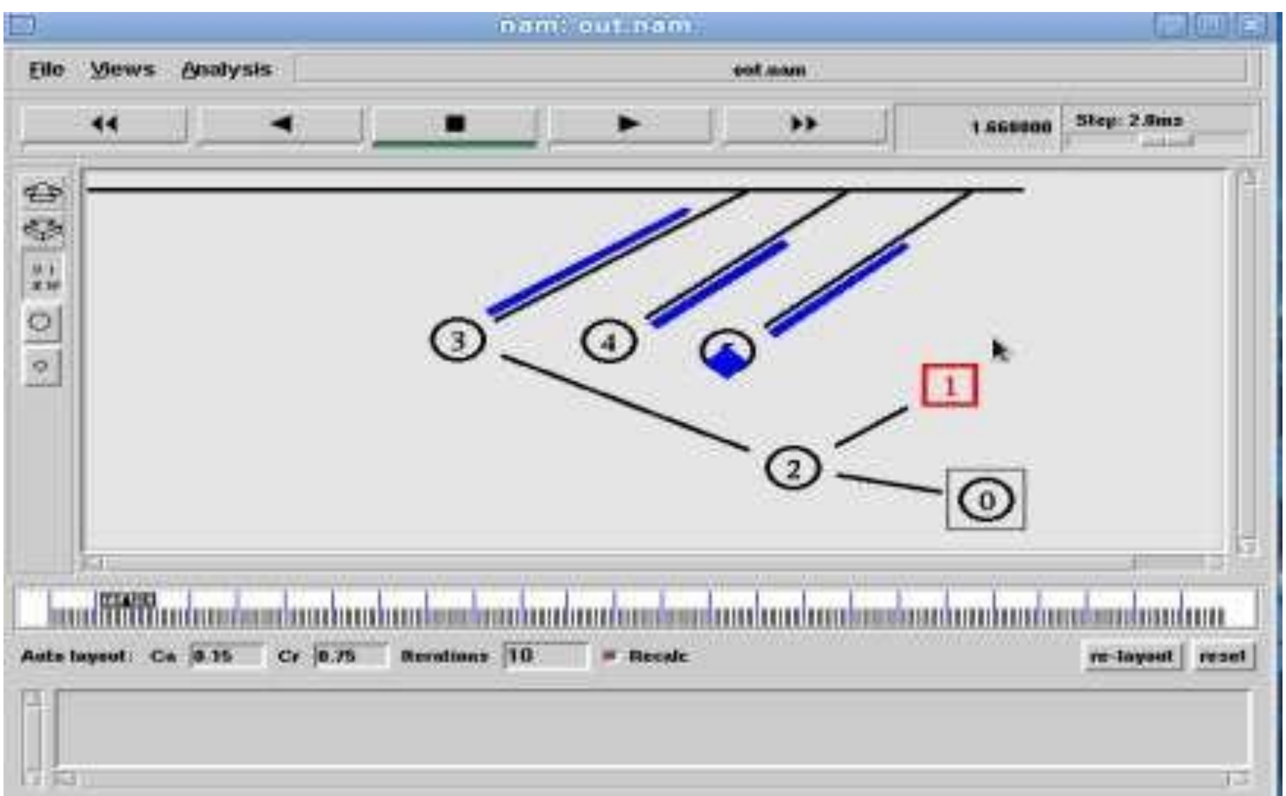
PROGRAM:

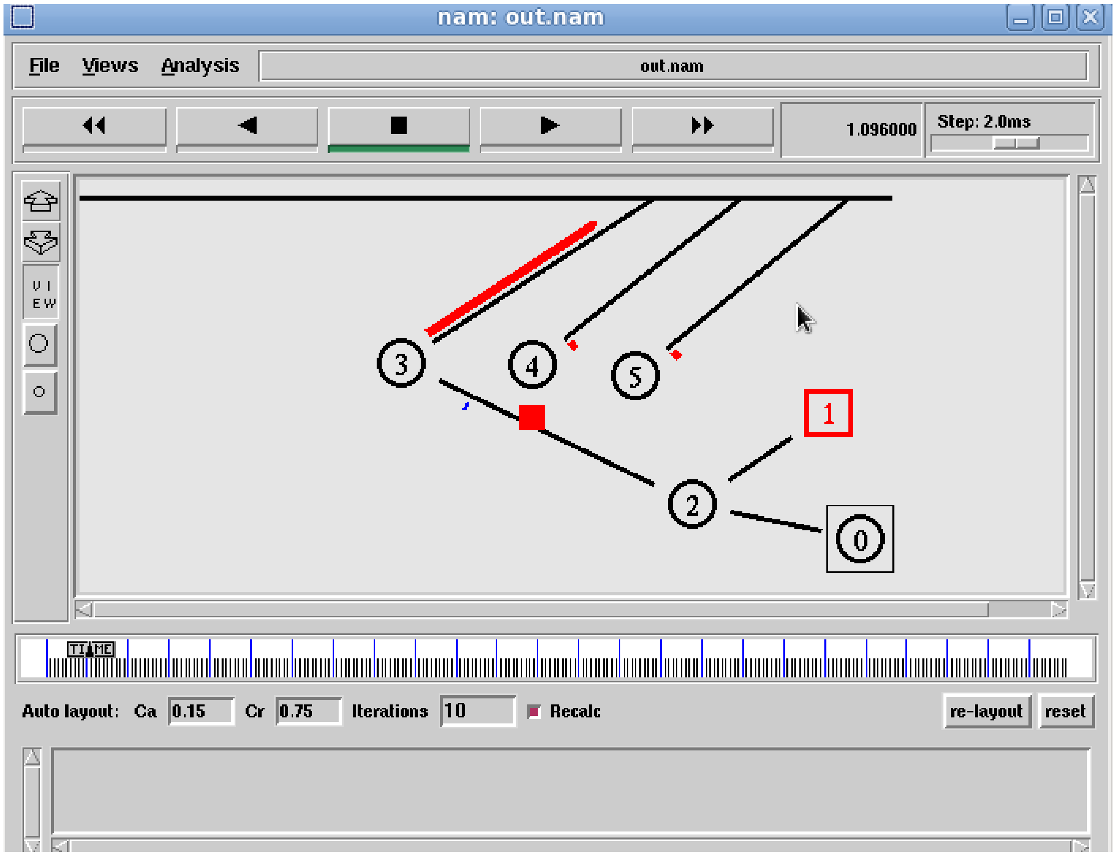
```
set ns [new Simulator]
$ns color 1 blue
$ns color 2 red
set fi1 [open out.tr w]
set winfile [open WinFile w]
$ns trace-all $fi1
set fi2 [open out.nam w]
$ns namtrace-all $fi2
proc finish {}
{
	global ns fi1 fi2
	$ns flush-trace
	close $fi1
	close $fi2
	exec nam out.nam &
	exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
 set n3 [$ns node]
 set n4 [$ns node]
 set n5 [$ns node]
$n1 color red
$n1 shape box
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
```

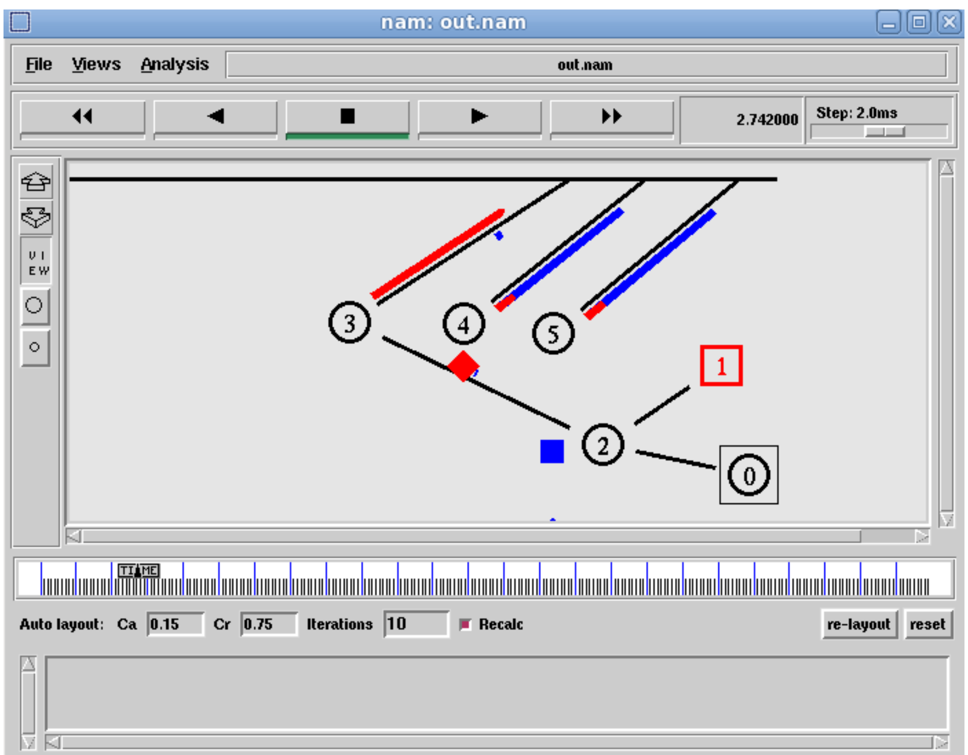
```
$tcp set fid_ 1
$tcp set window_ 8000
$tcp set packetsize_ 552
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.01mb
$cbr set random_ false
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 24.0 "$ftp stop"
$ns at 24.5 "$cbr stop"
proc plotwindow { tcpSource file }
{
	global ns
	set time 0.1

	set now [$ns now]
	set cwnd [$tcpSource set cwnd_]
	set wnd [$tcpSource set window_]
	puts $file "$now $cwnd"
	$ns at [expr $now+$time] "plotwindow $tcpSource $file"
}
$ns at 1.0 "plotwindow $tcp $winfile"
$ns at 5 "$ns trace-annotate \"packet drop\""
$ns at 125.0 "finish"
$ns run
```

OUTPUT:


nam: out.nam
File
Views
Analysis
out.nam
Step: 2.0ms
3
4
5
1
2
0
Auto layout:
Ca
0.15
Cr
0.75
Iterations
10
Recalc
re-layout
reset


nam: out.nam
File
Views
Analysis
out.nam
1.660000
Step: 2.0ms
3
4
1
2
0
Auto layout:
Ca
0.15
Cr
0.75
Iterations
10
Recalc
re-layout
reset





<u>RESULT</u>

Thus the carrier sense multiple access are implemented and executed successfully.