ARUNAI ENGINEERING COLLEGE

Velu Nagar, Tiruvannamalai-606603.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ACADEMIC YEAR : 2021-2022

Lab Manual

CS8581- Networks Laboratory (V semester)

Regulation 2017

Prepared By

Mrs.K.Sahitya Priyadharshini AP/CSE

Mrs.T.Lakshmi AP/CSE

INDEX

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

1. To afford the necessary background in the field of Information Technology to deal with engineering problems to excel as engineering professionals in industries.
2. To improve the qualities like creativity, leadership, teamwork and skill thus contributing towards the growth and development of society.
3. To develop ability among students towards innovation and entrepreneurship that caters to the needs of Industry and society.
4. To inculcate and attitude for life-long learning process through the use of information technology sources.
5. To prepare then to be innovative and ethical leaders, both in their chosen profession and in other activities.

PROGRAMME OUTCOMES (POs)

After going through the four years of study, Information Technology Graduates will exhibit ability to:

PO#	Graduate Attribute	Programme Outcome
1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2	Problem analysis	Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7	Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
10	Communication	Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions
11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
12	Life-long learning	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAM SPECIFIC OUTCOMES (PSOs)

By the completion of Information Technology program the student will have following Program specific outcomes

1. Design secured database applications involving planning, development and maintenance using state of the art methodologies based on ethical values.
2. Design and develop solutions for modern business environments coherent with the advanced technologies and tools.
3. Design, plan and setting up the network that is helpful for contemporary business environments using latest hardware components.
4. Planning and defining test activities by preparing test cases that can predict and correct errors ensuring a socially transformed product catering all technological needs.

CS8581 **NETWORKS LABORATORY** **L T P C 0 0 4 2**

OBJECTIVES

- ➢ To learn and use network commands.
- ➢ To learn socket programming.
- ➢ To implement and analyze various network protocols.
- ➢ To learn and use simulation tools.
- ➢ To use simulation tools to analyze the performance of various network protocols

LIST OF EXPERIMENTS

1. Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol analyzer andexamine.
2. Write a HTTP web client program to download a web page using TCP sockets.
3. Applications using TCP sockets like:
    - Echo client and echo server
    - Chat
    - 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.
10. Simulation of error correction code (like CRC).

TOTAL: 60 PERIODS

LIST OF EQUIPMENT FOR A BATCH OF 30 STUDENTS

HARDWARE

Standalone desktops	30 Nos

SOFTWARE:

- C / C++ / Java / Python / Equivalent Compiler 30 Nos.
- Network simulator like NS2/Glomosim/OPNET/ Packet Tracer / Equivalent

JAVA

Java is a high-level programming language originally developed by Sun Microsystems. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted, and Dynamic".

NS2

NS2 stands for Network Simulator Version 2. It is an open-source event-driven simulator designed specifically for research in computer communication networks. It provides substantial support to simulate bunch of protocols like TCP, FTP, UDP, https and DSR.It simulates wired and wireless network. It is primarily Unix based. Uses TCL as its scripting language.

COURSE OUTCOMES

CS8581.1	Implement various protocols using TCP and UDP.
CS8581.2	Compare the performance of different transport layer protocols.
CS8581.3	Use simulation tools to analyze the performance of various network protocols.
CS8581.4	Analyze various routing algorithms.
CS8581.5	Implement error correction codes.

CO-PO MATRIX

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CS8581.1	3		2	-	-	-	-	-	-	-	-	-
CS8581.2	3	1	2	-	-	-	-	-	-	-	-	-
CS8581.3				-	2	-	-	-	-	-	-	-
CS8581.4	2	1	2	-	-	-	-	-	-	-	-	-
CS8581.5	3	1	1	-	2	-	-	-	-	-	-	-
CS8581	3	1	2	-	2	-	-	-	-	-	-	-

CO-PSO MATRIX

COURSE	PSO 3
CS8581.1	**2**
CS8581.2	**3**
CS8581.3	**3**
CS8581.4	**3**
CS8581.5	**2**
CS8581	**3**

EVALUATION PROCEDURE FOR EACH EXPERIMENTS

S.No	Description	Mark
1.	Aim & Pre-Lab discussion	20
2.	Observation	20
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	20
Total		**100**

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Observation	05
2.	Performance	05
2.	Viva	05
3.	Record	05
Total		**20**

EX.NO:1 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.

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

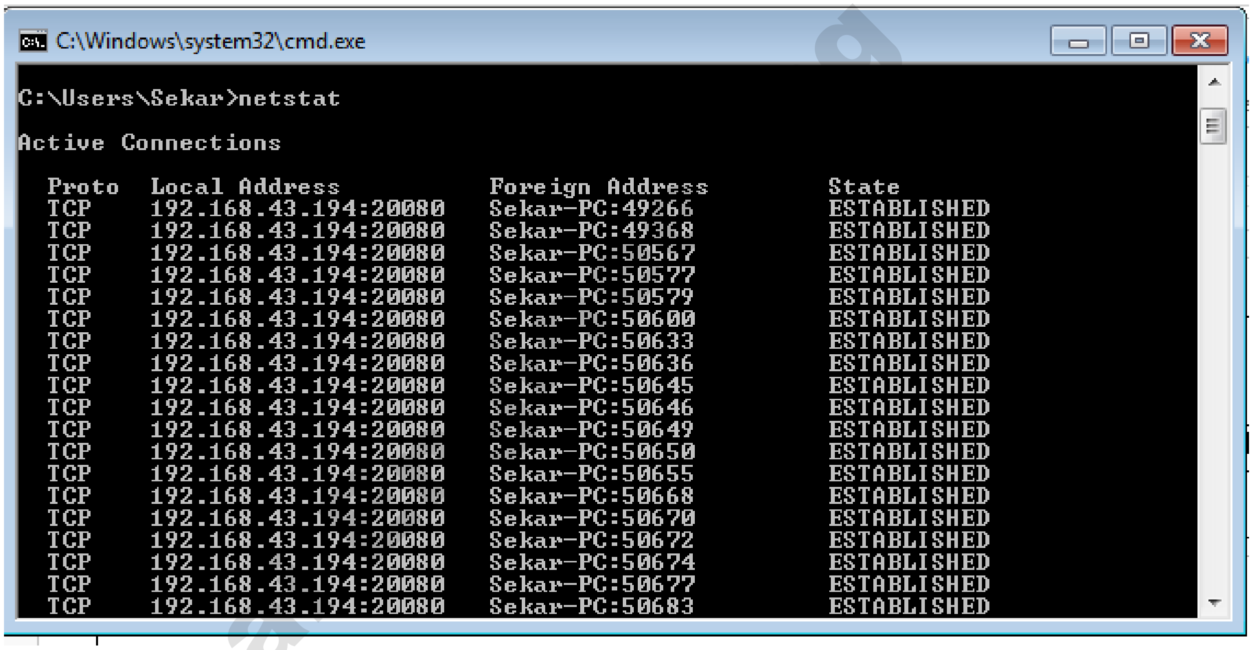
# tcpdump protocol
For example to display traffic only for the tcp traffic :
# tcpdump tcp
Filtering based on source or destination port
To filter based on a source or destination port:
# tcpdump src port ftp
# tcpdump dst port http

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


```
C:\Users\Sekar>netstat

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    192.168.43.194:20080   Sekar-PC:49266         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:49368         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50567         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50577         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50579         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50600         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50633         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50636         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50645         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50646         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50649         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50650         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50655         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50668         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50670         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50672         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50674         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50677         ESTABLISHED
  TCP    192.168.43.194:20080   Sekar-PC:50683         ESTABLISHED
```

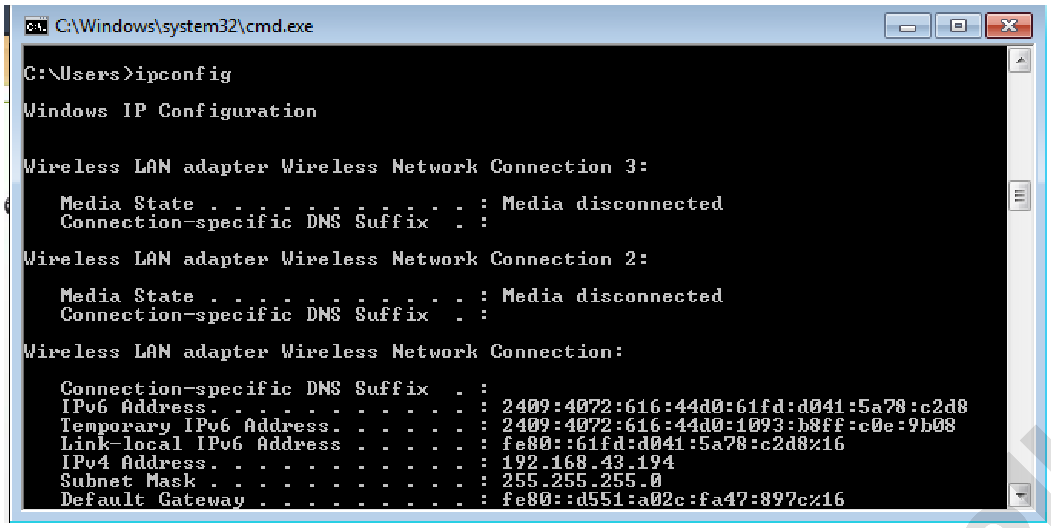
3. ipconfig

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



```
C:\Users>ipconfig

Windows IP Configuration


Wireless LAN adapter Wireless Network Connection 3:

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

Wireless LAN adapter Wireless Network Connection 2:

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

Wireless LAN adapter Wireless Network Connection:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : 2409:4072:616:44d0:61fd:d041:5a78:c2d8
   Temporary IPv6 Address. . . . . . : 2409:4072:616:44d0:1093:b8ff:c0e:9b08
   Link-local IPv6 Address . . . . . : fe80::61fd:d041:5a78:c2d8%16
   IPv4 Address. . . . . . . . . . . : 192.168.43.194
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::d551:a02c:fa47:897c%16
```

<u>4.nslookup</u>

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

<u>5.Trace route:</u>

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.

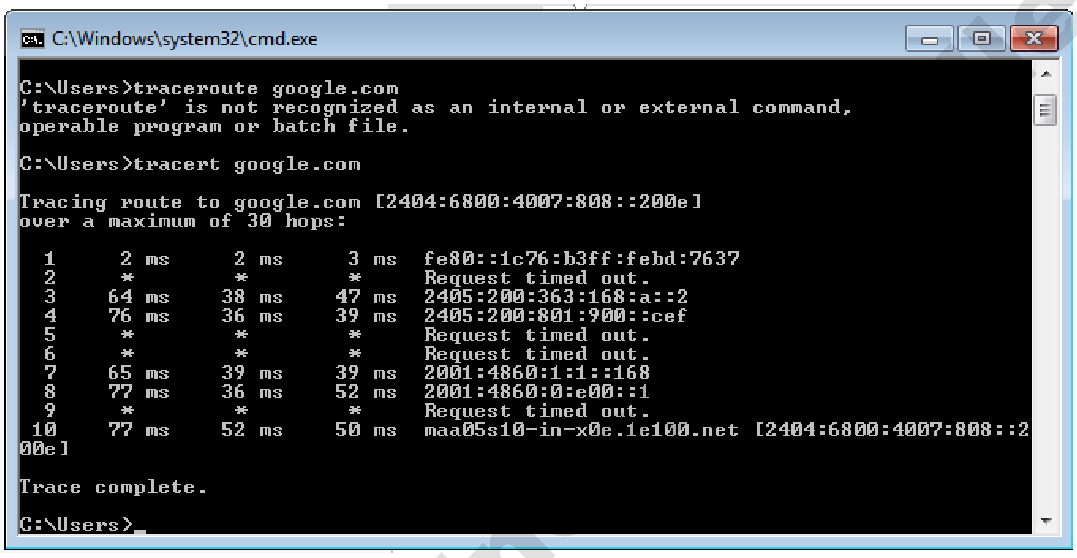
For the first set of packets, the first router receives the packet, decrements the TTL value and drops the packet because it then has TTL value zero. The router sends an ICMP Time Exceeded message back to the source. The next set of packets are given a TTL value of two, so the first router forwards the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



```
C:\Users>traceroute google.com
'traceroute' is not recognized as an internal or external command,
operable program or batch file.

C:\Users>tracert google.com

Tracing route to google.com [2404:6800:4007:808::200e]
over a maximum of 30 hops:

  1     2 ms     2 ms     3 ms  fe80::1c76:b3ff:febd:7637
  2     *        *        *     Request timed out.
  3    64 ms    38 ms    47 ms  2405:200:363:168:a::2
  4    76 ms    36 ms    39 ms  2405:200:801:900::cef
  5     *        *        *     Request timed out.
  6     *        *        *     Request timed out.
  7    65 ms    39 ms    39 ms  2001:4860:1:1::168
  8    77 ms    36 ms    52 ms  2001:4860:0:e00::1
  9     *        *        *     Request timed out.
 10    77 ms    52 ms    50 ms  maa05s10-in-x0e.1e100.net [2404:6800:4007:808::2
00e]

Trace complete.

C:\Users>_
```

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 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 .

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());
```

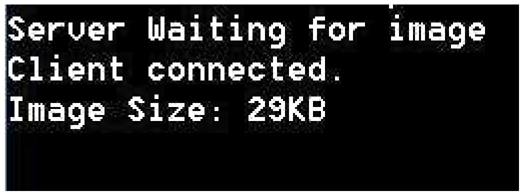
```
                    soc.close();
               }
               soc.close();
          }
}
```

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.



```
Server Waiting for image
Client connected.
Image Size: 29KB
```

RESULT:

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

Ex.No: 3 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

cho 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 listeningserver, 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 **Simulation of DNS using UDP Sockets**

AIM

To write a java program for DNS application

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 **Write a code simulating ARP /RARP protocols**

AIM:

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

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 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

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.

Program:

```
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 Study of TCP/UDP performance using Simulation tool.

AIM:

To simulate the performance of TCP/UDP using NS2.

Algorithm

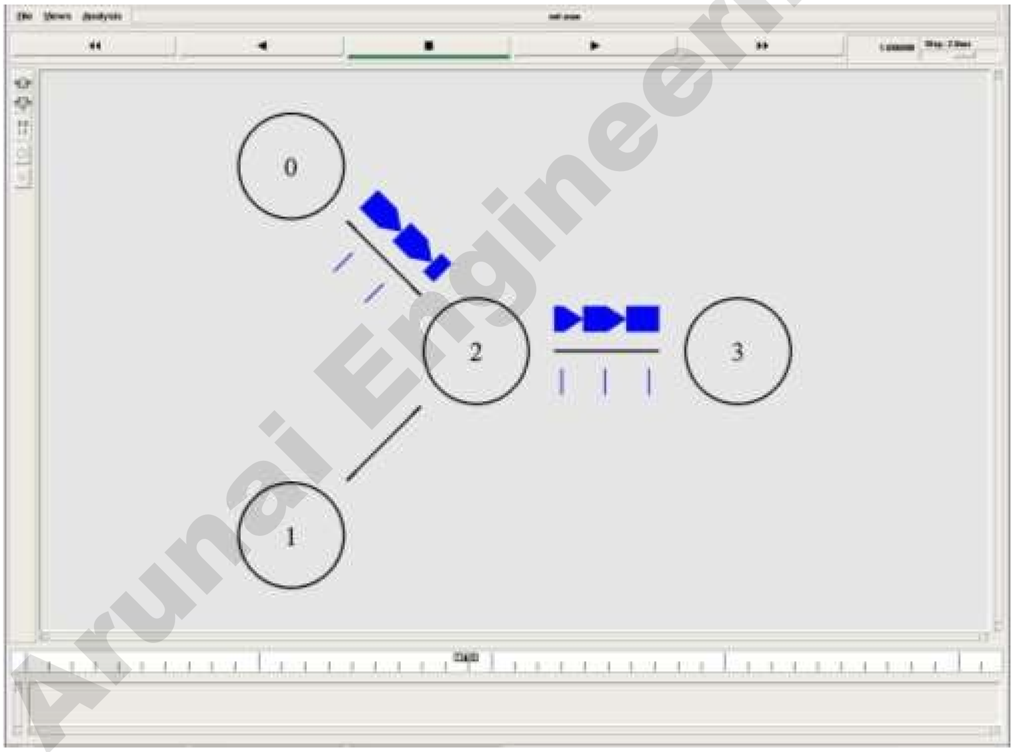
1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.
7. Create the applications and attach them to the tcp agent.
8. Connect tcp and tcp sink.
9. Run the simulation.

PROGRAM:

```
set ns [new Simulator]
$ns color 0 Blue
$ns color 1 Red
$ns color 2
Yellow set n0
[$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 0.5 set 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-
        trace close
        $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.

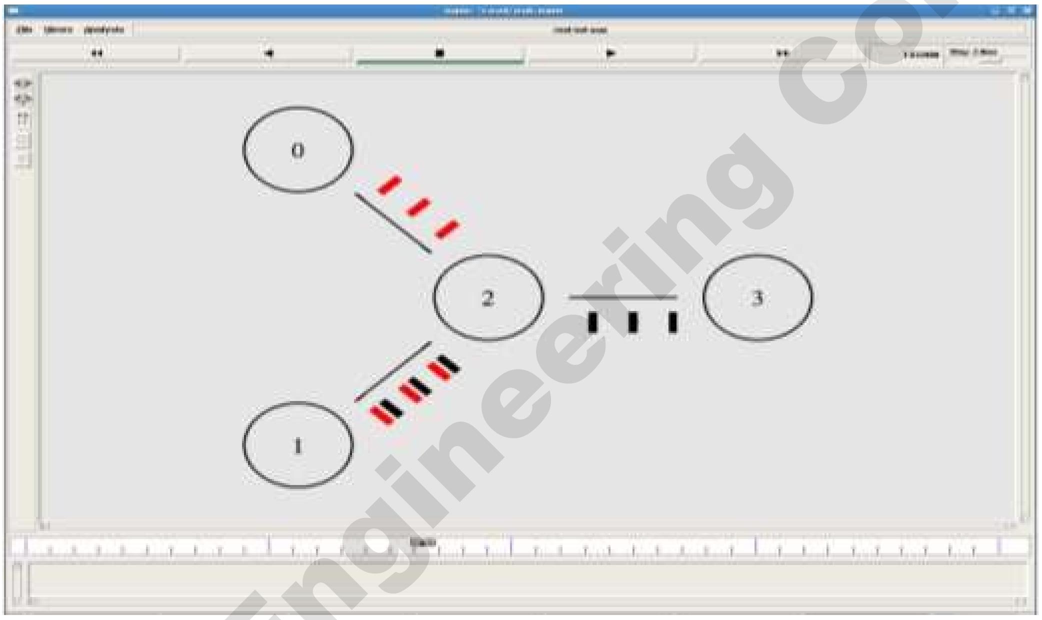
PROGRAM:

```
set ns [new Simulator]
$ns color 0 Blue
$ns color 1 Red
$ns color 2
Yellow set n0
[$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 0.5 set udp0
[newAgent/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
$null0 set 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-
        trace close
        $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 Simulation of Distance Vector/ Link State Routing algorithm.

AIM:

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

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.

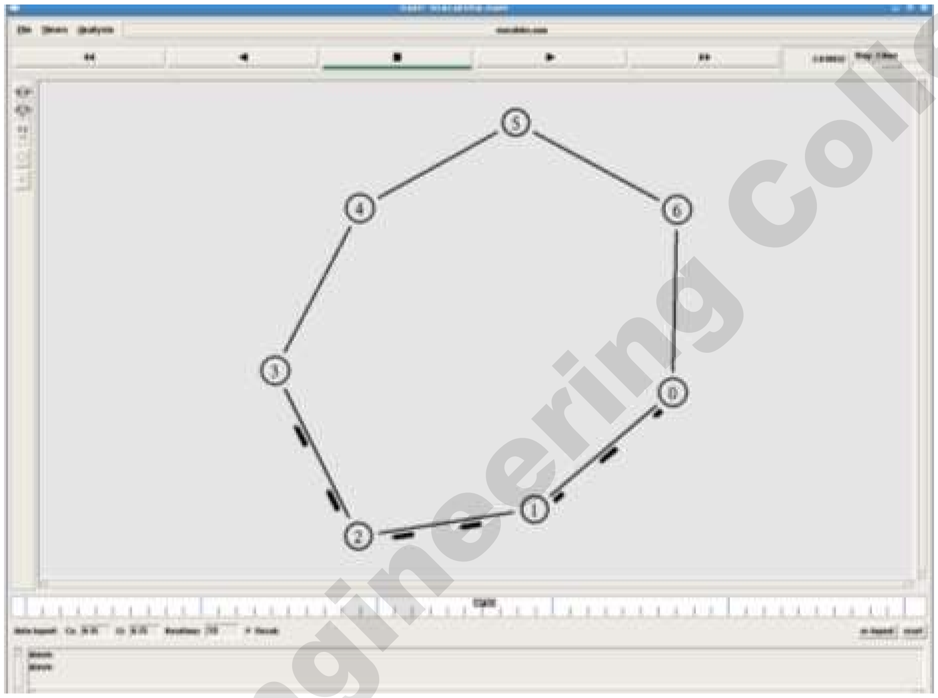
LINK STATE ROUTING PROTOCOL

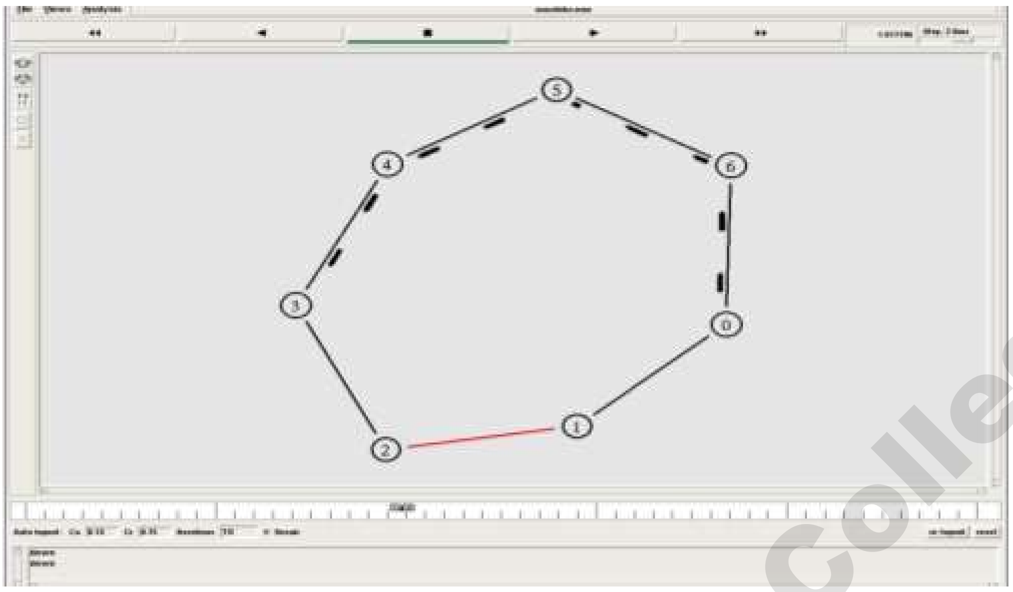
PROGRAM

```
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
$f0 proc
finish {} {
        global ns f0 nf
        $ns flush-trace
        close $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:




5
6
4
3
0
2
1

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

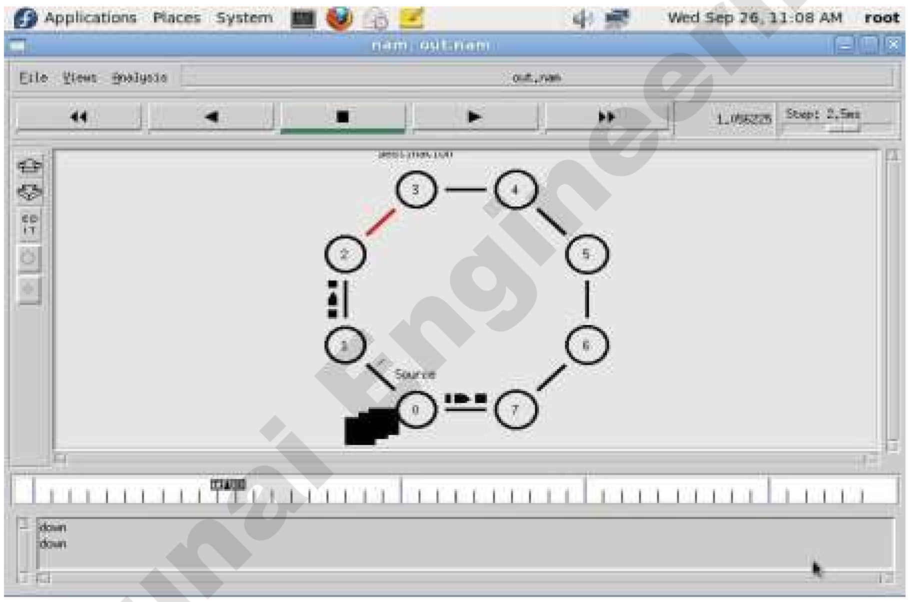
```
#Distance vector routing protocol – distvect.tcl
#Create a simulator object
set ns [new Simulator]
#Use distance vector routing
$ns rtproto DV
#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf
# Open tracefile
set nt [open trace.tr w]
$ns trace-all $nt
#Define 'finish' procedure

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]
```

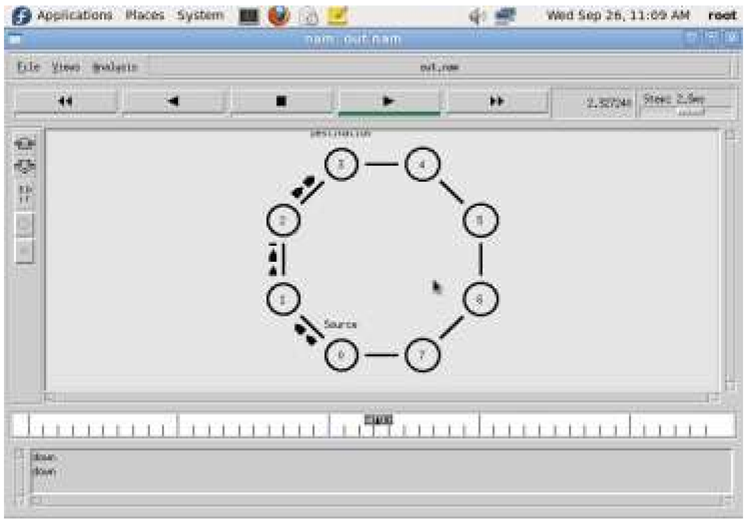
```
$ns attach-agent $n4 $null0
#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0
#Schedule events for the CBR agent and the network dynamics
$ns at 0.0 "$n1 label Source"
$ns at 0.0 "$n4 label Destination"
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n3 $n4
$ns rtmodel-at 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
```

OUTPUT

$ ns distvect.tcl



```
down
down
```



```
down
down
```

RESULT:

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

Ex.No:9 Performance Evaluation of Routing protocols using Simulation tool.

(a) UNICAST ROUTING PROTOCOL

AIM:

To write a ns2 program for implementing unicast routing protocol.

ALGORITHM:

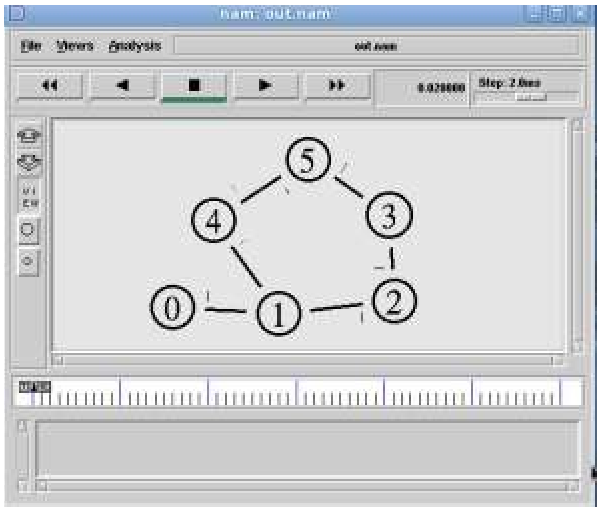
1. Start the program.
2. Declare the global variables ns for creating a new simulator.
3. Set the color for packets.
4. Open the network animator file in the name of file2 in the write mode.
5. Open the trace file in the name of file 1 in the write mode.
6. Set the unicast routing protocol to transfer 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.

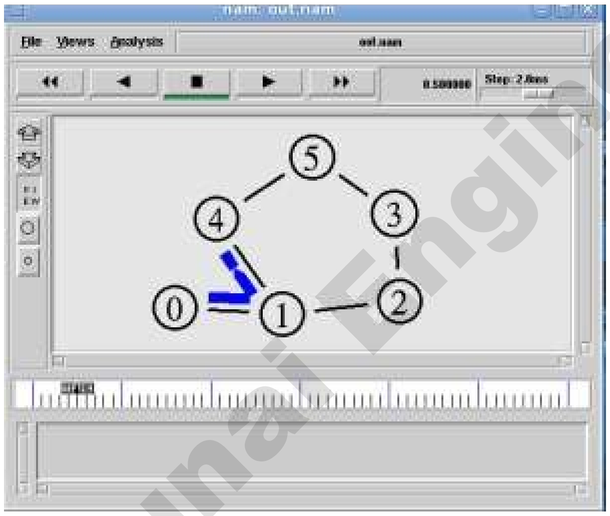
PROGRAM:

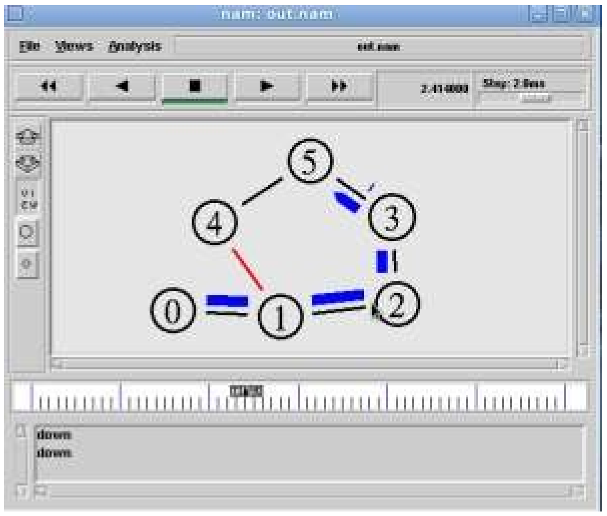
```
set ns [new Simulator]
#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
#Open the Trace file
set file1 [open out.tr w]
$ns trace-all $file1
#Open the NAM trace file
set file2 [open out.nam w]
$ns namtrace-all $file2
#Define a 'finish' procedure
proc finish {}
{
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







(b) 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 name file and execute the network animation file.
18. At the particular time call the finish procedure.
19. Stop the program.

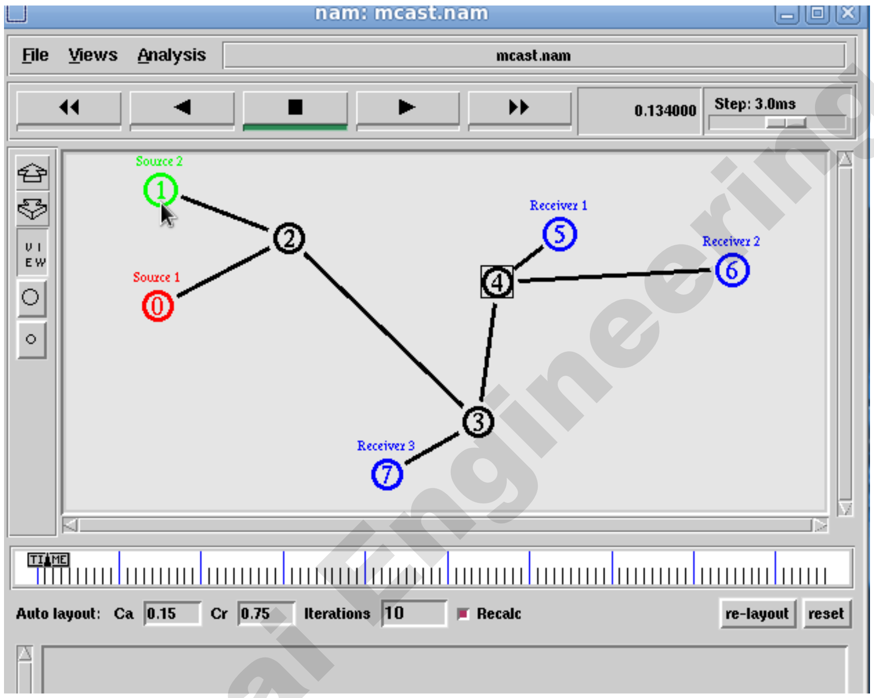
PROGRAM:

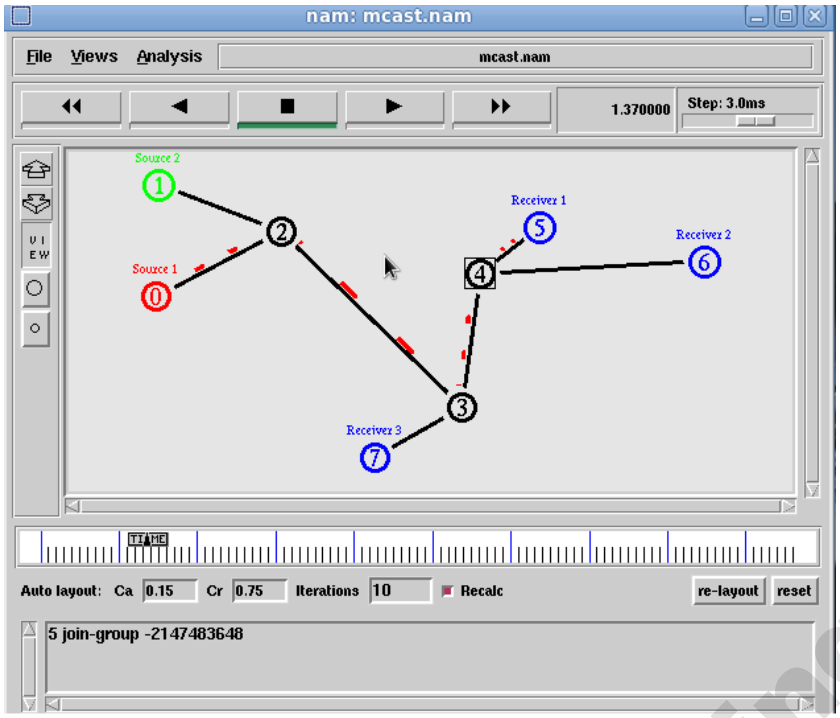
```
# Create scheduler
#Create an event scheduler wit multicast turned on
set ns [new Simulator -multicast on]
#$ns multicast
#Turn on Tracing
set tf [open output.tr w]
$ns trace-all $tf
# Turn on nam Tracing
set fd [open mcast.nam w]
$ns namtrace-all $fd
# Create nodes
set n0 [$ns node]
set n1 [$ns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

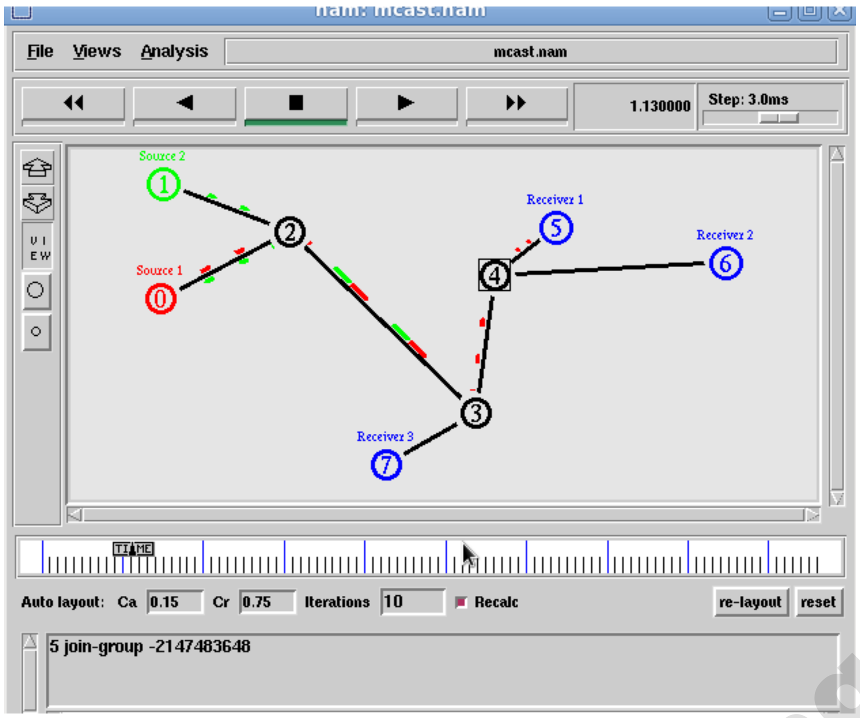
```
$n5 color blue
$n6 label "Receiver 2"
$n6 color blue
$n7 label "Receiver 3"
$n7 color blue

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

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





Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc

```
5 join-group -2147483648
```



5 join-group -2147483648

RESULT:

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

Ex.No:10 Simulation of Error Detection Code (like CRC)

AIM:

To implement error checking code using java.

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.