TAGORE INSTITUTE OF ENGINEERING AND TECHNOLOGY

Deviyakurichi-636112, Attur (TK), Salem (DT). Website: www.tagoreiet.ac.in

(Approved by AICTE, New Delhi and Affiliated to Anna University, Chennai)

Accredited by NAAC

Department of Computer Science and Engineering

III Year- V Semester – CSE

CS8581 Networks Laboratory

LAB MANUAL

Academic Year 2020-2021
(2017 Regulation)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

V SEM – CSE

CS8581 NETWORKS LABORATORY

S.NO	NAME OF EXPERIMENTS
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 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).

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

V SEM – CSE

CS8581 NETWORKS LABORATORY

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 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).

EX.NO:1 **COMMANDS**

AIM:

Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol analyzer and examine.

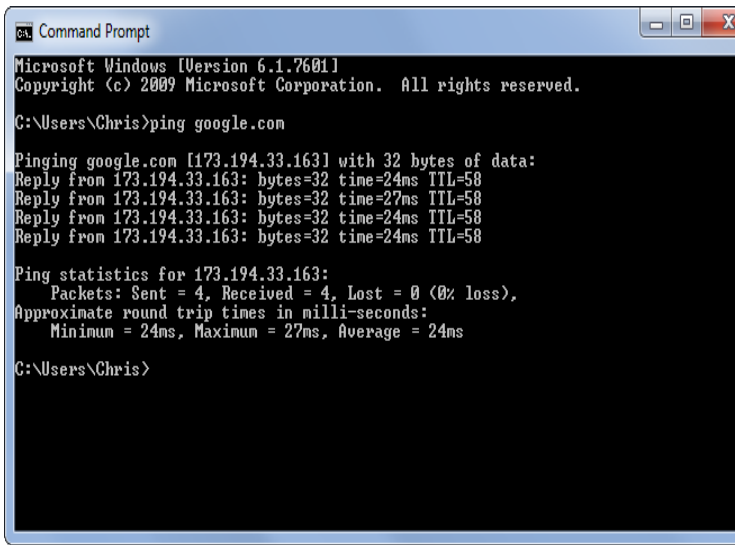
1.ping

RELATED: How To Troubleshoot Internet Connection Problems

The ping command sends ICMP echo request packets to a destination. For example, you could run

ping google.com or **ping 173.194.33.174** to ping a domain name or IP address.

These packets ask the remote destination to reply. If the remote destination is configured to reply, it
will respond with packets of its own. You'll be able to see how long the round-trip time is between
your computer and the destination. You'll see a "request timed out" message if packet loss is occurring, and you'll see an error message if your computer can't communicate with the remote host at all.
This tool can help you troubleshoot Internet connection problems, but bear in mind that many servers
and devices are configured not to reply to pings.

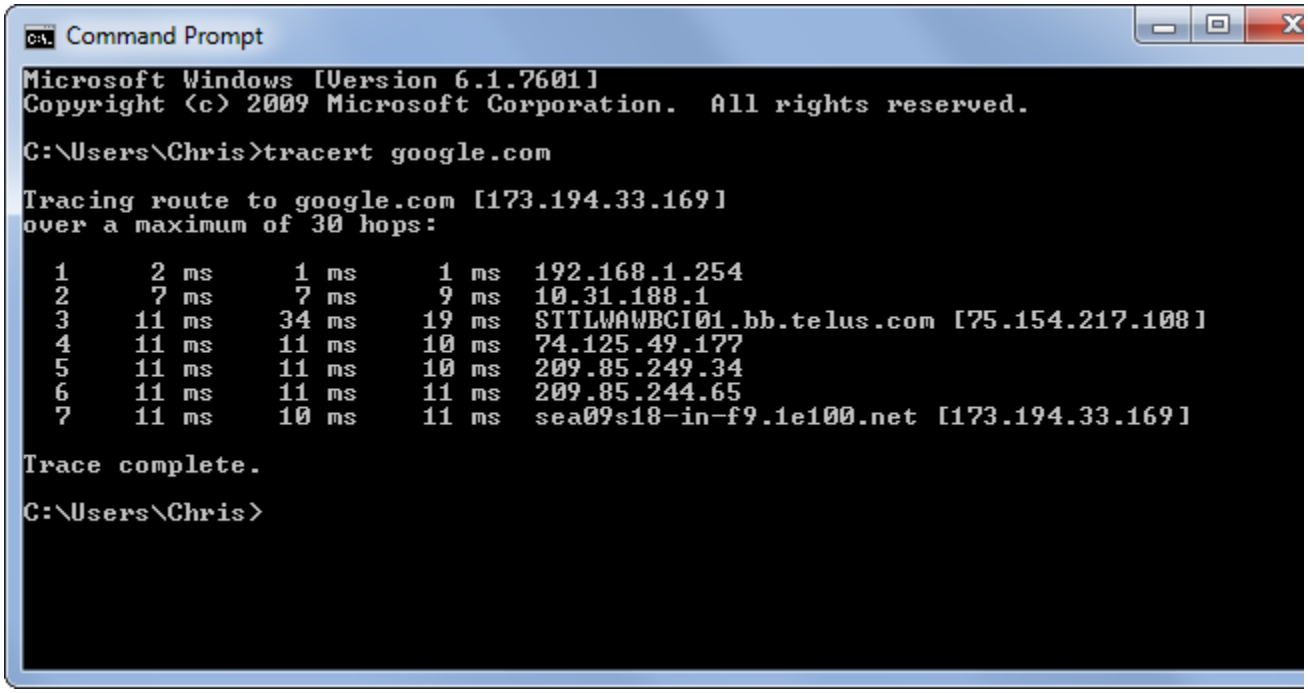


2.traceroute / tracert / tracepath

The traceroute, tracert, or tracepath command is similar to ping, but provides information about the path a packet takes. traceroute sends packets to a destination, asking each Internet router

along the way to reply when it passes on the packet. This will show you the path packets take when you send them between your location and a destination.

This tool can help troubleshoot connection problems. For example, if you can't communicate with a server, running traceroute may show you where the problem is occurring between your computer and the remote host.



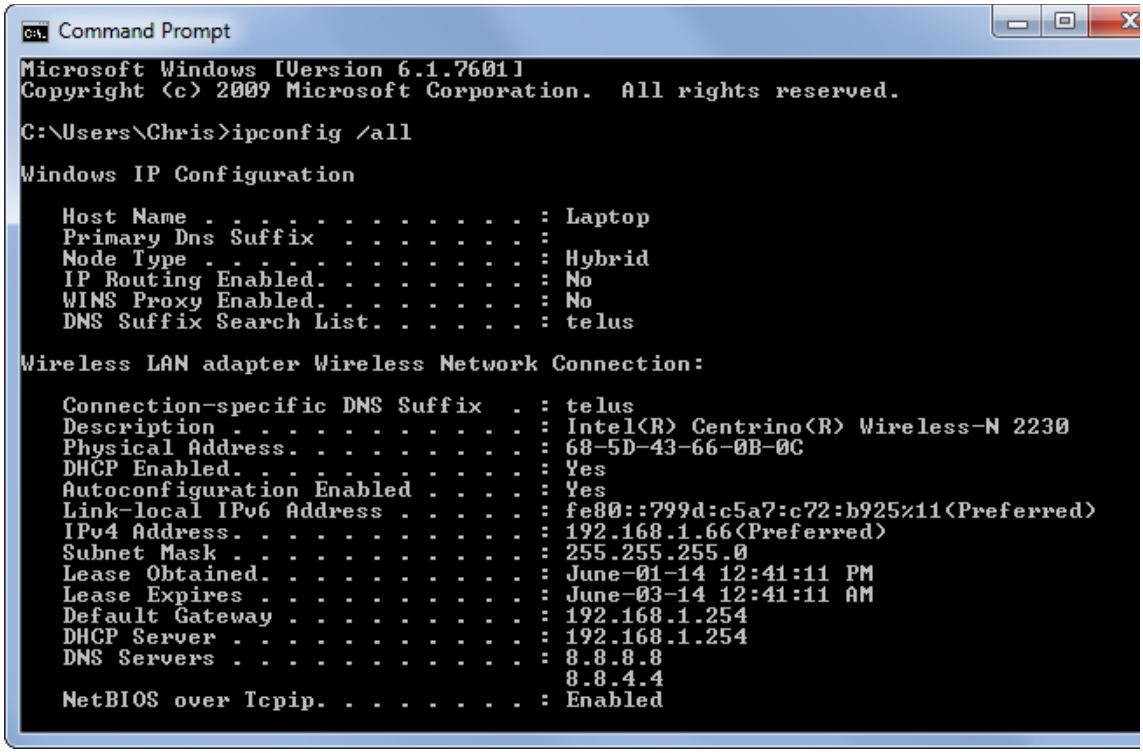
3.ipconfig / ifconfig

RELATED: 10 Useful Windows Commands You Should Know

The ipconfig command is used on Windows, while the ifconfig command is used on Linux, Mac OS X,

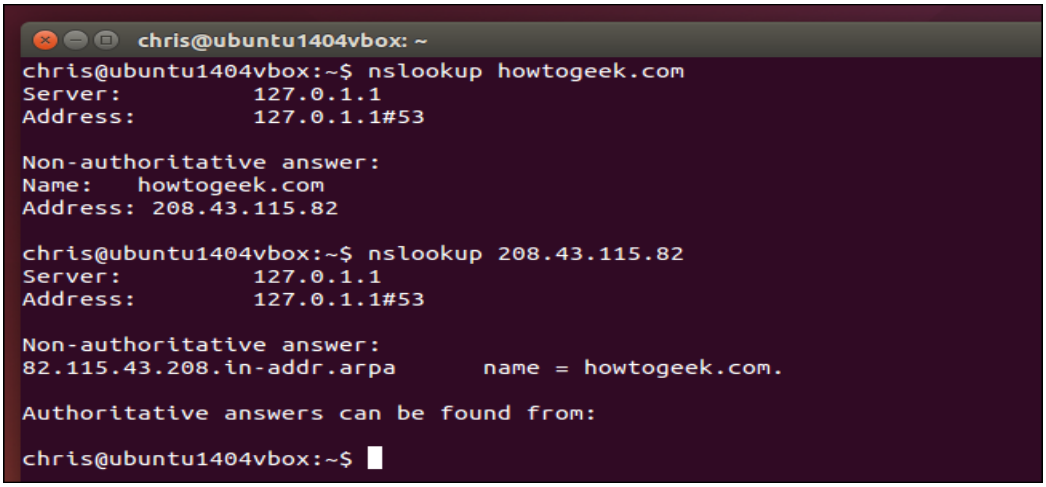
and other Unix-like operating systems. These commands allow you to configure your network interfaces and view information about them.

For example, you can use the ipconfig /all command on Windows to view all your configured network interfaces, their IP addresses, DNS servers, and other information. Or, you can use the ipconfig /flushdns command to flush your DNS cache, forcing Windows to get new addresses from its DNS servers every time you contact a new hostname. Other commands can force your computer to release its IP address and get a new one from its DHCP server. This utility can quickly display your computer's IP address or help you troubleshoot problems.



4.nslookup

The nslookup command will look up the IP addresses associated with a domain name.
nslookup also allows you to perform a reverse lookup to find the domain name associated with an IP address. For example, **nslookup 208.43.115.82** will show you that this IP address is associated with howtogeek.com.



Netstat is a useful tool for checking network and Internet connections. Some useful applications for the average PC user are considered, including checking for malware connections.

Syntax and switches

The command syntax is netstat [-a] [-b] [-e] [-f] [-n] [-o] [-p proto] [-r] [-s] [-t] [-v] [interval] A brief description of the switches is given in Table I below. Some switches are only in certain Windows versions, as noted in the table..Note that switches for Netstat use the dash symbol "-" rather than the slash "/".

Switch	Description
-a	Displays all connections and listening ports
-b	Displays the executable involved in creating each connection or listening port. (Added in XP SP2.)
-e	Displays Ethernet statistics
-f	Displays Fully Qualified Domain Names for foreign addresses. (In Windows Vista/7 only)
-n	Displays addresses and port numbers in numerical form
-o	Displays the owning process ID associated with each connection
-p proto	Shows connections for the protocol specified by proto; proto may be any of: TCP, UDP, TCPv6, or UDPv6.
-r	Displays the routing table
-s	Displays per-protocol statistics
-t	Displays the current connection offload state, (Windows Vista/7)
-v	When used in conjunction with -b, will display sequence of components involved in creating the connection or listening port for all executables. (Windows XP SP2, SP3)
[interval]	An integer used to display results multiple times with specified number of seconds between displays. Continues until stopped by command ctrl+c. Default setting is to display once,

Applications of Netstat

Netstat is one of a number of command-line tools available to check the functioning of a network. (See this page for discussion of other tools.) It provides a way to check if various aspects of TCP/IP are working and what connections are present. In Windows XP SP2, a new switch "-B" was added that allows the actual executable file that has opened a connection to be displayed. This newer capability provides a chance to catch malware that may be phoning

home or using your computer in unwanted ways on the Internet. There are various ways that a system administrator might use the assortment of switches but I will give two examples that might be useful to home PC users.

Checking TCP/IP connections

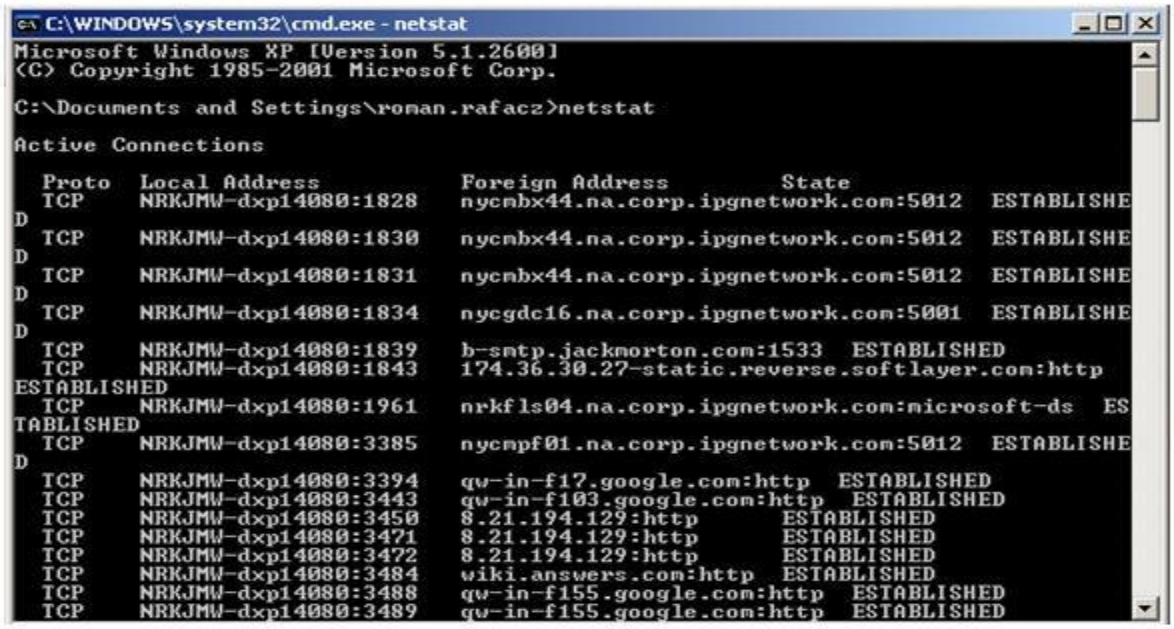
TCP and UDP connections and their IP and port addresses can be seen by entering a command combining two switches: netstat -an An example of the output that is obtained is shown in Figure 1.

The information that is displayed includes the protocol, the local address, the remote (foreign) address, and the connection state. Note that the various IP addresses include port information as well. An explanation of the different connection states is given in Table II>

State	Description
CLOSED	Indicates that the server has received an ACK signal from the client and the connection is closed
CLOSE_WAIT	Indicates that the server has received the first FIN signal from the client and the connection is in the process of being closed
ESTABLISHED	Indicates that the server received the SYN signal from the client and the session is established
FIN_WAIT_1	Indicates that the connection is still active but not currently being used
FIN_WAIT_2	Indicates that the client just received acknowledgment of the first FIN signal from the server
LAST_ACK	Indicates that the server is in the process of sending its own FIN signal
LISTENING	Indicates that the server is ready to accept a connection
SYN_RECEIVED	Indicates that the server just received a SYN signal from the client
SYN_SEND	Indicates that this particular connection is open and active
TIME_WAIT	Indicates that the client recognizes the connection as still active but not currently being used

Checking for malware by looking at which programs initiate connections

To find out which programs are making connections with the outside world, we can use the command netstat -b (Note that for Windows Vista/7, this particular switch requires that the command prompt have elevated privileges.) Actually, it is better to check over a period of time and we can add a number that sets the command to run at fixed intervals. Also, it is best to create a written record of the connections that are made over some period of time. The command can then be written netstat -b 5 >> C:\connections.txt Note that as written, this command will run with five-second intervals until stopped by entering "Ctrl+c", which is a general command to exit. (Some reports say that this can be fairly CPU intensive so it may cause a slower, single-core machine to run sluggishly. It was not noticeable on my dual-core machine.) A simple example of the type of output is shown in Figure 2. Note that the Process ID (PID) is given when using Windows XP. In Windows Vista/7, the switch "o' has to be added to display PIDs. This command can be combined with other tools such as Task Manager to analyze what executable files and processes are active and are trying to make Internet connections.



6. TCPDUMP

tcpdump command is also called as packet analyzer.

tcpdump command will work on most flavors of unix operating system. tcpdump allows us to save the packets that are captured, so that we can use it for future analysis. The saved file can be viewed by the same tcpdump command. We can also use open source software like wireshark to read the tcpdump pcap files.

In this tcpdump tutorial, let us discuss some practical examples on how to use the tcpdump command.

1. Capture packets from a particular ethernet interface using tcpdump -i

When you execute tcpdump command without any option, it will capture all the packets flowing through all the interfaces. -i option with tcpdump command, allows you to filter on a particular ethernet interface.

```
$ tcpdump -i eth1
14:59:26.608728 IP xx.domain.netbcp.net.52497 > valh4.lell.net.ssh: . ack 540 win 16554
14:59:26.610602 IP resolver.lell.net.domain > valh4.lell.net.24151:  4278 1/0/0 (73)
14:59:26.611262 IP valh4.lell.net.38527 > resolver.lell.net.domain: 26364+ PTR? 244.207.104.10.in-addr.arpa. (45)
```

In this example, tcpdump captured all the packets flows in the interface eth1 and displays in the standard output.

Result:

Thus the commands tcpdump, netstat, ifconfig, nslookup and traceroute are studied.

EX.NO 2. CREATE A SOCKET FOR HTTP FOR WEB PAGE UPLOAD AND DOWNLOAD.

Aim:

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

Algorithm

1.Start the program.
2.Get the frame size from the user
3.To create the frame based on the user request.
4.To send frames to server from the client side.
5.If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
6.Stop the program

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 for HTTP for web page upload and download was created.

EX-NO 3. **APPLICATIONS USINGTCP SOCKETS**

A. ECHO CLIENT AND ECHO SERVER

Aim

To write a java program for application using TCPSockets Links.

Algorithm

1.Start the program.

2.Get the frame size from the user

3.To create the framebased on the user request.

4.To send frames to server from the client side.

5.If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.

6.Stop the 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();

                }
                catch(IOException e)
                {
                        System.out.println(e);
                }
                try
                {
                        os=new PrintStream(c.getOutputStream());

is=new DataInputStream(System.in);

DataInputStream(c.getInputStream());

{
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
Note: EServer.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
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
Note: EClient.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
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

Aim

Write a Program client –server application for chat using UDP Sockets

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

Program

File Client

```
import   java.io.*;   import   java.net.*;
import java.util.*; class Clientfile
{           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 file name:");
String str=in.readLine(); dout.writeBytes(str+'\n');
System.out.println("Enter the new file name:"); String str2=in.readLine();
String str1,ss;
FileWriter f=new FileWriter(str2); char buffer[];
while(true)
{ str1=din.readLine(); if(str1.equals("-1")) break;
System.out.println(str1); buffer=new char[str1.length()];
str1.getChars(0,str1.length(),buffer,0);
f.write(buffer);
  }
f.close();
clsct.close();
}
catch (Exception e)
{
 System.out.println(e);
 }}}
```

Server

```
import java.io.*; import java.net.*; import java.util.*; class Serverfile
{ public static void main(String args[])
{
Try
{
 ServerSocket obj=new ServerSocket(139);
 while(true)
{
 Socket obj1=obj.accept();
DataInputStream din=new DataInputStream(obj1.getInputStream()); DataOutputStream dout=new
```

```
DataOutputStream(obj1.getOutputStream()); String str=din.readLine();
FileReader f=new FileReader(str);
BufferedReader b=new BufferedReader(f);
String s;
while((s=b.readLine())!=null) { System.out.println(s);
dout.writeBytes(s+'\n');
 }
f.close(); dout.writeBytes("-1\n");
 }            }
catch(Exception e)
{            System.out.println(e);}
 }
}
```

Output

File content
Computer networks jhfcgsauf

jbsdava jbvuesagv client

Enter the file name: sample.txt

server

Computer networks jhfcgsauf

jbsdava jbvuesagv **client**

Enter the new file name: net.txt

Computer networks jhfcgsauf

jbsdava jbvuesagv Destination file

Computer networks jhfcgsauf

Jbsdavajbvuesagv

Result:

Thus the applications using TCP sockets for echo client and echo server, chat, file transfer was created.

EX-NO 4. SIMULATION OF DNS USING UDP SOCKETS

Aim

To write a java program for DNS application

Algorithm

1.Start the program.
2.Get the frame size from the user
3.To create the frame based on the user request. 4.To send frames to server from the client side.
5.If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
6.Stop the program

Program

/ UDP DNS Server
Udpdnsserver

```
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 –** Udpdnsclient

```
.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();
}
}
```

OUTPU
T Server

```
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 Simulation of DNS using UDP Sockets was created.

EX.NO:5 WRITE A CODE SIMULATING ARP /RARP PROTOCOLS.

Aim:

To write a java program for simulating ARP protocols using TCP

ALGORITHM:

Client

1. Start the program
2. Using socket connection is established between client and server.
3. Get the IP address to be converted into MAC address.
4. Send this IP address to server.
5. Server returns the MAC address to client.

Server

1. Start the program
2. Accept the socket which is created by the client.
3. Server maintains the table in which IP and corresponding MAC addresses are stored.
4. Read the IP address which is send by the client.
5. Map the IP address with its MAC address and return the MAC address to client.

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

EX.NO (5(b) PROGRAM FOR REVERSE ADDRESS RESOLUTION PROTOCOL (RARP) USING UDP

Aim:

To write a java program for simulating RARP protocols using UDP

ALGORITHM:

Client

1.Start the program
2. using datagram sockets UDP function is established.
2.Get the MAC address to be converted into IP address.
3.Send this MAC address to server.
4.Server returns the IP address to client.

Server

1. Start the program.
2. Server maintains the table in which IP and corresponding MAC addresses are stored.
3. Read the MAC address which is send by the client.
4. Map the IP address with its MAC address and return the IP address to client.

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=newDatagramPacket(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 Simulation for simulating ARP/RARP protocols using UDP was created.

EX-NO 6. STUDY OF NETWORK SIMULATOR (NS) AND SIMULATION OF CONGESTION CONTROL ALGORITHMS USING NS

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

INTRODUCTION:

A device, computer program or system used during software verification, which behaves or operates like a given system when provided with a set of controlled inputs.

NEED OF SIMULATORS:

- Customers with the simulator spend less time debugging simple program errors
- The simulator makes it easy to write and test code
- It is easier for our support engineers to explain complex problems if you have a simulator.
- The simulator requires no setup time. An emulator may require configuration and a target board before you can debug.

EXAMPLES OF SIMULATORS:

Some of the famous network simulators available are listed below-

- NS-2
- OPNET
- PADNS
- GTNETS
- GLOMOSIM
- M5-SIMULATOR
- DARMOUTH SSF

BACKGROUND OF NS:

NS began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. Currently ns development is support through DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI

NS doesn't give priority to GUI fundamentally and is almost a text-based code. Thus, users should write the code by Text Editor. But since NS is free software, it has been used widely by many researchers and its debugging is freely done. Therefore, the Source Codes of the current NS are very reliable and are configured systematically. Nowadays, many people have been using NS for Network Simulators

NS-2 NETWORK SIMULATOR:

NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkley written in C++ and OTcl. NS is primarily useful for simulating local and wide area networks. NS2 is an open-source simulation tool that runs on Linux Although NS is fairly easy to use once you get to know the simulator; it is quite difficult for a first time user, because there are few user-friendly manuals. Even though there is a lot of documentation written by the developers, which has in depth explanation of the simulator, it is written with the depth of a skilled NS user.

NETWORK ANIMATOR (NAM)

NS together with its companion, NAM, form a very powerful set of tools for teaching networking concepts. NS contains all the IP protocols typically covered in undergraduate and most graduate courses, and many experimental protocols contributed by its ever-expanding users base. With nam, these protocols can visualized as animations.

NAM GRAPHICAL EDITOR

This is the latest (Sept. 2001) addition to NAM. With this editor, you no longer have to type TCL code to create animations. You can create your network topology and simulate various protocols and traffic sources by dragging the mouse.

What can you do with these tools?

Create	Visualize
▪ Terrestrial, satellite and wireless networks with various routing algorithms (DV, LS, PIM-DM, PIM-SM, AODV, DSR). ▪ Traffic sources like web, ftp, telnet, cbr, and stochastic traffic. ▪ Failures, including deterministic, probabilistic loss, link failure, etc. ▪ Various queuing disciplines (drop-tail, RED, FQ, SFQ, DRR, etc.) and QoS .	▪ Packet flow, queue builds up and packet drops. ▪ Protocol behavior: TCP slow start, self-clocking, congestion control, fast retransmit and recovery. ▪ Node movement in wireless networks. ▪ Annotations to highlight important events. ▪ Protocol state (e.g., TCP cwnd).

OVERVIEW OF NS

The overall simulation procedure in the NS is shown in Fig.1. First of all, there is a language named OTcl, and this is an object oriented version language which was converted from the conventional Tcl (Tool Command Language). As shown in Fig.2, an OTcl Script representing a given network configuration is written. NS is simply composed of OTcl Script and OTcl Interpreter. OTcl Interpreter can translate the code related to NS using NS Simulation Library.

NS simulation results can be observed through graphs for analysis or animations with NAM (Network Animator). NAM is a network animator that is developed for education purposes. It can display the detailed procedure of the simulation. NAM is not used in this experiment.

Very simply speaking, NS is the Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network set-up module libraries. Users write an OTcl script using OTcl.

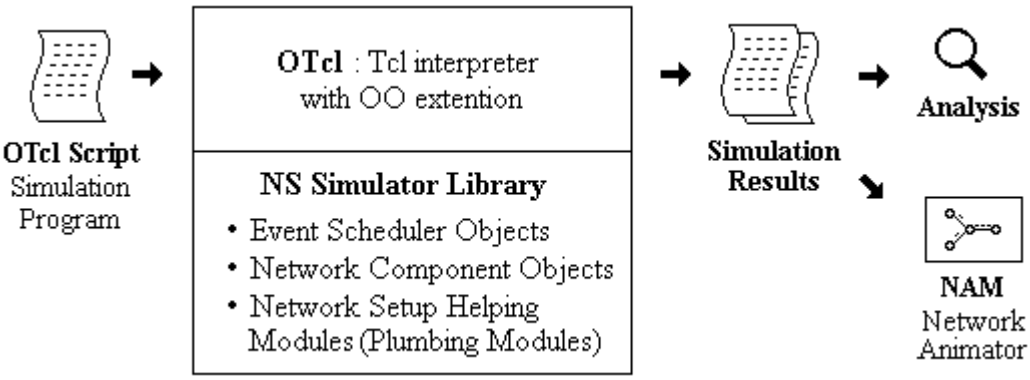


Fig.1 Simplified User's View of NS

(1) Duality of OTcl and C++

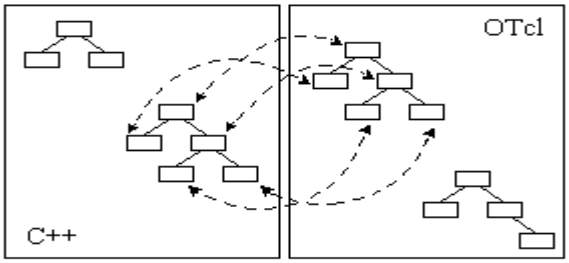


Fig.2 C++ and OTcl: The Duality

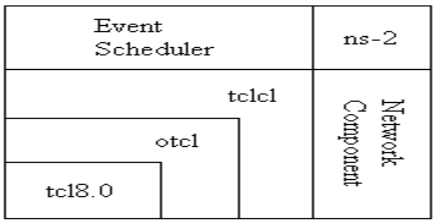


Figure 3. Architectural View of NS

As shown in Fig.3, NS has a one-to-one mapping structure between C++ and OTcl. For example, if there exists a class which implements WFQ (Weighted Fair Queuing) in C++, there exists a similar Class in OTcl. This structure is introduced for improving the simulation speed and providing the convenience. And the core components in network simulation, such as Node, Link, and Queue are written using C++ Classes. Since OTcl performs at a slow processing speed, the components for Network Simulation are written using C++ Classes, and the process that comprises Network by connecting these components with each other is written in OTcl Script. Moreover, NS is implemented using a dual structure as shown in the above figure for accessing components written in C++ Classes through OTcl Script.

Consequently, if a user makes a new Network Component, he/she should write C++ Classes and OTcl Classes and the part to connect each other. Since it is somewhat difficult to write C++ Classes and to perform Simulation, it is omitted in this experiment.

(2) NS DIRECTORY STRUCTURE

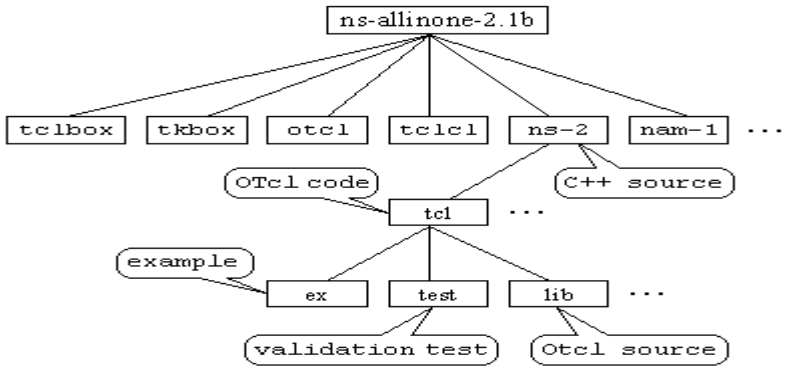


Fig.4 NS Directory Structure

Among the sub-directories of ns-allinone-2.1b shown in Fig.4, ns-2 has all of the simulator implementations (either in C++ or in OTcl), validation test OTcl scripts and example OTcl scripts. Within this

directory, all OTcl codes and test/example scripts are located under the sub-directory called tcl, and most of C++ code.

For example, NS's 2.1b has the Directory of ns-allinone-2.1b, which is usually located in /usr/local/ns-allinone-2.1b. All kinds of tools needed in NS form the sub-directory structure, and in particular, ns-2 directory has all source codes of C++ Class. Since every Network Component is written in C++ Class in this way, users can easily understand the real operations of Network Components to be used. This is one of large advantages.

There is also a tcl directory in ns-2 directory. The tcl directory has OTcl Codes, and its sub-directories include several examples, validation test, and OTcl sources.

(3) LEARNING OTCL

The first step to utilize NS is to understand the OTcl language. Fig.5 shows a program example written in Tcl language. In Tcl, the keyword 'proc' is used to define a procedure, followed by a procedure name and arguments in the first brace and definition in the second brace. At this moment, note that you must use the brace, differently from C++. Note that you'll encounter an error if you move the last brace "{" of "proc test {} {" to the next line.

If you run "set a 43", NS checks whether or not the variable "a" is already declared. If the variable name has already been declared, NS inserts 43 into

```
# Writing a procedure called "test"
proc test {} {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
        if {$k < 5} {
            puts "k < 5, pow = [expr pow($d, $k)]"
        } else {
            puts "k >= 5, mod = [expr $d % $k]"
        }
    }
}

# Calling the "test" procedure created above
test
```

Fig.5 A Sample Tcl Script

"a". Otherwise, a new one is declared and NS substitutes 43 into it. In other words, declaration and assignment are combined together. "set a" means that the instance "a" is declared only in case that "a" doesn't exist. In "set c [expr $a + $b]", one thing to note is that to get the value assigned to a variable, $ is used with the variable name, and another thing is that "expr" is used to do mathematical computations. Therefore, 43+27=70 is inserted into c. When we use "for" command, we initialize k with "{set k 0}", and give a condition with "{$k < 10}", and update for Index variable with "{incr k}". Finally, the keyword 'puts' is to print out the following string within double quotation marks like "printf()" in C.

OTcl

OTcl is the language which modifies the conventional Tcl to an object-oriented version. Fig.6 shows an example of an OTcl script that defines two object classes, "mom" and "kid", where "kid" is the child class of "mom", and a member function called "greet" for each class. After defining the classes, each object instance is declared, the "age" variable of each instance is set to 45 (for mom) and 15 (for kid), and the "greet" member function of each object instance is called.

```
# add a member function call "greet"
Class mom
mom instproc greet {} {
     $self instvar age_
     puts "$age_ year old mom say:
     How are you doing?"
}

# Create a child class of "mom" called "kid"
# and overide the member function "greet"
Class kid -superclass mom
kid instproc greet {} {
     $self instvar age_
     puts "$age_ year old kid say:
     What's up, dude?"
}

# Create a mom and a kid object, set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15

# Calling member function "greet" of each object
$a greet
$b greet
```

Fig.6 A Sample OTcl Script

The keyword 'Class' is to create an object class and 'instproc' is to define a member function to an object class. Class inheritance is specified using the keyword '-superclass.' In defining member functions, '$self' acts like the "this" pointer in C++, and 'instvar' is used to check if the following variable name has already been declared in its class or in its superclass. If the variable name has already been declared, the variable is referred. Otherwise, a new one is declared. Finally, to create an object instance, the keyword 'new' is used as shown in the example.

(4) NS CLASS HIERARCHY

If you understood OTcl in some extent, you need to look into the Class Hierarchy of NS. NS has the Class Hierarchy which is implemented using both C++ and OTcl, as shown in Fig.7.

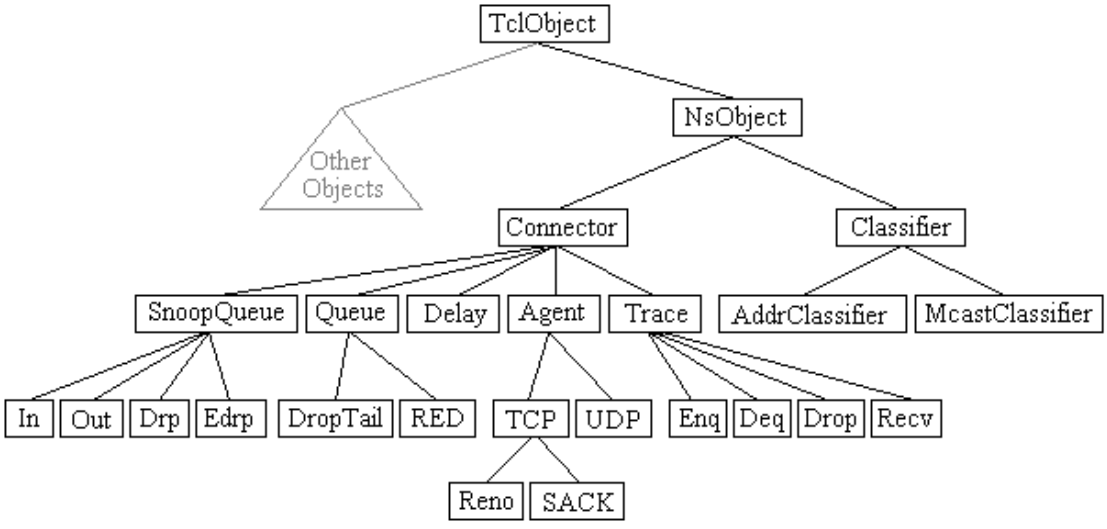


Fig.7 NS Class Hierarchy

The root of the hierarchy is the TclObject class that is the superclass of all OTcl libraries. As a

descendant class of TclObject, NsObject class is the superclass of all basic network component objects. Fig. 17 shows the NS Class Hierarchy. Actually, it has a very complex class hierarchy. In the figure, AQM (Active Queue Management) such as DropTail, RED, REM, and DRR is implemented under Queue. TCP means TCP Tahoe. TCP Reno, Newreno, and SACK are implemented by inheriting TCP Tahoe.

Now you can write OTcl Script to perform a simulation by using Network Components of NS. I would recommend that you perform some simple simulations by referring to the front part of NS Manual [6]. I also would recommend that you refer to very useful examples of OTcl Script located in /usr/local/ns-allinone-xxx/ns-xxx/tcl/ex.

HARDWARE/SOFTWARE REQUIREMENTS

To build ns you need a computer and a C++ compiler. The all-in-one package requires about 320MB of disk space to build. Building ns from pieces can save some disk space. There are two ways to build ns: from all the pieces or all at once. If you just want to try it out quickly, you might try all at once. If you want to do C-level development, or save download time or disk space, or have trouble with all-in-one you should build it from the pieces

Ns-allinone is a package, which contains enquired components and some optional components used in running ns. The package contains an "install" script to automatically configure, compile and install these components. After downloading, run the install script. If you haven't installed ns before and want to quickly try ns out, ns-allinone may be easier than getting all the pieces by hand.

Result:

Thus the study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS was done.

EX NO 7.a STUDY OF UDP PERFORMANCE

AIM:

To study the performance comparison of User Datagram Protocol.

INTRODUCTION:

The User Datagram Protocol (UDP) is one of the core members of the Internet Protocol Suite, the set of network protocols used for the Internet. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths. UDP is sometimes called the Universal Datagram Protocol. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP uses a simple transmission model without implicit hand-shaking dialogues for guaranteeing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers).Common network applications that use UDP include: the Domain Name System (DNS), streaming media applications such as IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and many online games.

TCP AND UDP PORT:

UDP applications use datagram sockets to establish host-to-host communications. Sockets bind the application to service ports, that function as the endpoints of data transmission. A port is a software structure that is identified by the port number, a 16 bit integer value, allowing for port numbers between 0 and 65,535. Port 0 is reserved, but is a permissible source port value if the sending process does not expect messages in responsse.The Internet Assigned Numbers Authority has divided port numbers into three ranges.

PACKET STRUCTURE:

UDP is a minimal message-oriented Transport Layer protocol that is documented in IETF RFC 768.UDP provides no guarantees to the upper layer protocol for message delivery and the UDP protocol layer retains no state of UDP messages once sent. For this reason, UDP is sometimes referred to as Unreliable Datagram Protocol.UDP provides application multiplexing (via port numbers) and integrity verification (via checksum) of the header and payload.If transmission reliability is desired, it must be implemented in the user's application.

bits 0 – 15 16 – 31

0 Source Port Number Destination Port Number

32 Length Checksum

64

Data

The UDP header consists of 4 fields, all of which are 2 bytes (16 bits).

RELIABILITY AND CONGESTION CONTROL SOLUTIONS :

Lacking reliability, UDP applications must generally be willing to accept some loss, errors or duplication. Some applications such as TFTP may add rudimentary reliability mechanisms into the application layer as needed.[2] Most often, UDP applications do not require reliability mechanisms and may even be

hindered by them. Streaming media, real-time multiplayer games and voice over IP (VoIP) are examples of applications that often use UDP. If an application requires a high degree of reliability, a protocol such as the Transmission Control Protocol or erasure codes may be used instead.

Lacking any congestion avoidance and control mechanisms, network-based mechanisms are required to minimize potential congestion collapse effects of uncontrolled, high rate UDP traffic loads. In other words, since UDP senders cannot detect congestion, network-based elements such as routers using packet queuing and dropping techniques will often be the only tool available to slow down excessive UDP traffic. The Datagram Congestion Control Protocol (DCCP) is being designed as a partial solution to this potential problem by adding end host TCP-friendly congestion control behavior to high-rate UDP streams such as streaming media.

APPLICATIONS:

Numerous key Internet applications use UDP, including: the Domain Name System (DNS), where queries must be fast and only consist of a single request followed by a single reply packet, the Simple Network Management Protocol (SNMP), the Routing Information Protocol (RIP)[1] and the Dynamic Host Configuration Protocol (DHCP).

Voice and video traffic is generally transmitted using UDP. Real-time video and audio streaming protocols are designed to handle occasional lost packets, so only slight degradation in quality occurs, rather than large delays if lost packets were retransmitted. Because both TCP and UDP run over the same network, many businesses are finding that a recent increase in UDP traffic from these real-time applications is hindering the performance of applications using TCP, such as point of sale, accounting, and database systems. When TCP detects packet loss, it will throttle back its data rate usage. Since both real-time and business applications are important to businesses, developing quality of service solutions is seen as crucial by some.

COMPARISON OF UDP AND TCP :

Transmission Control Protocol is a connection-oriented protocol, which means that it requires handshaking to set up end-to-end communications. Once a connection is set up user data may be sent bi-directionally over the connection.

Reliable – TCP manages message acknowledgment, retransmission and timeout. Multiple attempts to deliver the message are made. If it gets lost along the way, the server will re-request the lost part. In TCP, there's either no missing data, or, in case of multiple timeouts, the connection is dropped.

Ordered – if two messages are sent over a connection in sequence, the first message will reach the receiving application first. When data segments arrive in the wrong order, TCP buffers the out-of-order data until all data can be properly re-ordered and delivered to the application.

Heavyweight – TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.

Streaming – Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.

UDP is a simpler message-based connectionless protocol. Connectionless protocols do not set up a dedicated end-to-end connection. Communication is achieved by transmitting information in one direction from source to destination without verifying the readiness or state of the receiver.

Unreliable – When a message is sent, it cannot be known if it will reach its destination; it could get lost along the way. There is no concept of acknowledgment, retransmission or timeout.

Not ordered – If two messages are sent to the same recipient, the order in which they arrive cannot be predicted.

Lightweight – There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.

Datagrams – Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield

an entire message as it was originally sent.

IMPLEMENTATION
ALGORITHM
Step 1: Define different color for data flows for NAM
Step 2: Open the NAM trace file
Step 3: Define a finished procedure
Step 4: Define a desired Number of nodes
Step 5: Create a links between the nodes
Step 6: Specifies the position of the node for NAM
Step 7: Set the queue size between desired nodes
Step 8: Setup a udpConnection
Step 9: Setup a FTP over udp Connection
Step 10: Define a procedure for plotting window size
Step 11: Specify the end of simulation

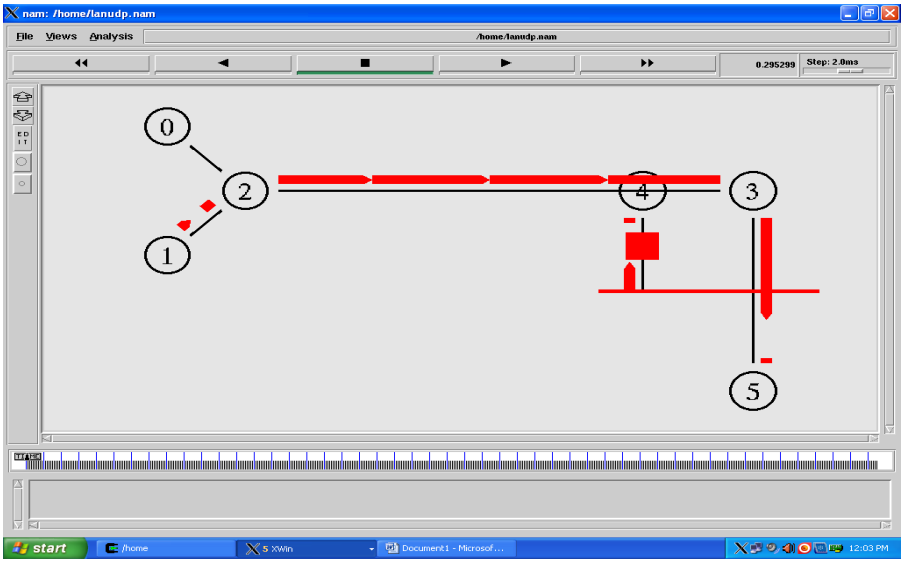
PROGRAM FOR UDP

```
set ns [new Simulator]

$ns color 1 Blue
$ns color 2 Red

set tracefile1 [open lanudp.tr w]
set winfile [open WinFile w]
$ns trace-all $tracefile1

set namfile [open lanudp.nam w]
$ns namtrace-all $namfile

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$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 "$n4 $n5 $n3" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n3 $n2 orient left
```

```
$ns queue-limit $n2 $n3 20

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 packetSize_ 1000
$cbr set rate_ 0.01Mb
$cbr set random_ false

$ns at 0.1 "$cbr start"

$ns at 124.5 "$cbr  stop"

proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file"
}
$ns run
```

OUTPUT:

UDP



EX. NO: 7 b **STUDY OF TCP PERFORMANCE**
DATE:
AIM:

To study the performance comparison of Transmission Control Protocol.

INTRODUCTION:

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite (the other being Internet Protocol, or IP), so the entire suite is commonly referred to as TCP/IP.

TCP SEGMENT STRUCTURE:

A TCP segment consists of a segment header and a data section. The TCP header contains 10 mandatory fields, and an optional extension field (Options, pink background in table).

The data section follows the header. Its contents are the payload data carried for the application. The length of the data section is not specified in the TCP segment header. It can be calculated by subtracting the combined length of the TCP header and the encapsulating IP segment header from the total IP segment length (specified in the IP segment header).

Protocol operation:

A Simplified TCP State Diagram. See TCP EFSM diagram for a more detailed state diagram including the states inside the ESTABLISHED state.TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (connection establishment) before entering the data transfer phase. After data transmission is completed, the connection termination closes established virtual circuits and releases all allocated resources.

A TCP connection is managed by an operating system through a programming interface that represents the local end-point for communications, the Internet socket. During the lifetime of a TCP connection it undergoes a series of state changes:

LISTEN : In case of a server, waiting for a connection request from any remote client.

SYN-SENT : waiting for the remote peer to send back a TCP segment with the SYN and ACK flags set. (usually set by TCP clients)

SYN-RECEIVED : waiting for the remote peer to send back an acknowledgment after having sent back a connection acknowledgment to the remote peer. (usually set by TCP servers)

ESTABLISHED : the port is ready to receive/send data from/to the remote peer.

FIN-WAIT-1

FIN-WAIT-2

CLOSE-WAIT

CLOSING

LAST-ACK

TIME-WAIT : represents waiting for enough time to pass to be sure the remote peer received the acknowledgment of its connection termination request. According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes.

CLOSED

Connection establishment:

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

The active open is performed by the client sending a SYN to the server. It sets the segment's sequence number

to a random value A.

In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number (A + 1), and the sequence number that the server chooses for the packet is another random number, B.

Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. A + 1, and the acknowledgement number is set to one more than the received sequence number i.e. B + 1.

At this point, both the client and server have received an acknowledgment of the connection.

MAXIMUM SEGMENT SIZE:

The Maximum segment size (MSS) is the largest amount of data, specified in bytes, that TCP is willing to send in a single segment. For best performance, the MSS should be set small enough to avoid IP fragmentation, which can lead to excessive retransmissions if there is packet loss. To try to accomplish this, typically the MSS is negotiated using the MSS option when the TCP connection is estabslished, in which case it is determined by the maximum transmission unit (MTU) size of the data link layer of the networks to which the sender and receiver are directly attached. Furthermore, TCP senders can use Path MTU discovery to infer the minimum MTU along the network path between the sender and receiver, and use this to dynamically adjust the MSS to avoid IP fragmentation within the network.

CONNECTION TERMINATION :

The connection termination phase uses, at most, a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. Therefore, a typical tear-down requires a pair of FIN and ACK segments from each TCP endpoint.

A connection can be "half-open", in which case one side has terminated its end, but the other has not. The side that has terminated can no longer send any data into or receive any data from the connection,

but the other side can (but generally if it tries, this should result in no acknowledgment and therefore a timeout, or else result in a positive RST, and either way thereby the destruction of the half-open socket).

It is also possible to terminate the connection by a 3-way handshake, when host A sends a FIN and host B replies with a FIN & ACK (merely combines 2 steps into one) and host A replies with an ACK.[13] This is perhaps the most common method.

HARDWARE IMPLEMENTATIONS :

One way to overcome the processing power requirements of TCP is to build hardware implementations of it, widely known as TCP Offload Engines (TOE). The main problem of TOEs is that they are hard to integrate into computing systems, requiring extensive changes in the operating system of the computer or device. One company to develop such a device was Alacritech.

PERFORMANCE COMPARISON :

For many applications TCP is not appropriate. One big problem (at least with normal implementations) is that the application cannot get at the packets coming after a lost packet until the retransmitted copy of the lost packet is received. This causes problems for real-time applications such as streaming multimedia (such as Internet radio), real-time multiplayer games and voice over IP (VoIP) where it is sometimes more useful to get most of the data in a timely fashion than it is to get all of the data in order.

For both historical and performance reasons, most storage area networks (SANs) prefer to use Fibre Channel protocol (FCP) instead of TCP/IP.

Also for embedded systems, network booting and servers that serve simple requests from huge numbers of clients (e.g. DNS servers) the complexity of TCP can be a problem. Finally some tricks such as transmitting data between two hosts that are both behind NAT (using STUN or similar systems) are far simpler without a relatively complex protocol like TCP in the way.

Generally where TCP is unsuitable the User Datagram Protocol (UDP) is used. This provides the application multiplexing and checksums that TCP does, but does not handle building streams or

retransmission giving the application developer the ability to code those in a way suitable for the situation and/or to replace them with other methods like forward error correction or interpolation.

SCTP is another IP protocol that provides reliable stream oriented services not so dissimilar from TCP. It is newer and considerably more complex than TCP, and has not yet seen widespread deployment. However, it is especially designed to be used in situations where reliability and near-real-time considerations are important.

Venturi Transport Protocol (VTP) is a patented proprietary protocol that is designed to replace TCP transparently to overcome perceived inefficiencies related to wireless data transport.

TCP also has issues in high bandwidth environments. The TCP congestion avoidance algorithm works very well for ad-hoc environments where the data sender is not known in advance, but if the environment is predictable, a timing based protocol such as Asynchronous Transfer Mode (ATM) can avoid TCP's retransmits overhead.

Multipurpose Transaction Protocol (MTP/IP) is patented proprietary software that is designed to adaptively achieve high throughput and transaction performance in a wide variety of network conditions, particularly those where TCP is perceived to be inefficient.

TCP OVER WIRELESS NETWORKS:

TCP has been optimized for wired networks. Any packet loss is considered to be the result of congestion and the congestion window size is reduced dramatically as a precaution. However, wireless links are known to experience sporadic and usually temporary losses due to fading, shadowing, hand off, and other radio effects, that cannot be considered congestion. After the (erroneous) back-off of the congestion window size, due to wireless packet loss, there can be a congestion avoidance phase with a conservative decrease in window size. This causes the radio link to be underutilized. Extensive research has been done on the subject of how to combat these harmful effects. Suggested solutions can be categorized as end-to-end solutions (which require modifications at the client and/or server), link layer solutions (such as RLP in CDMA2000), or proxy based solutions (which require some changes in the network without modifying end nodes).

IMPLEMENTATION

ALGORITHM

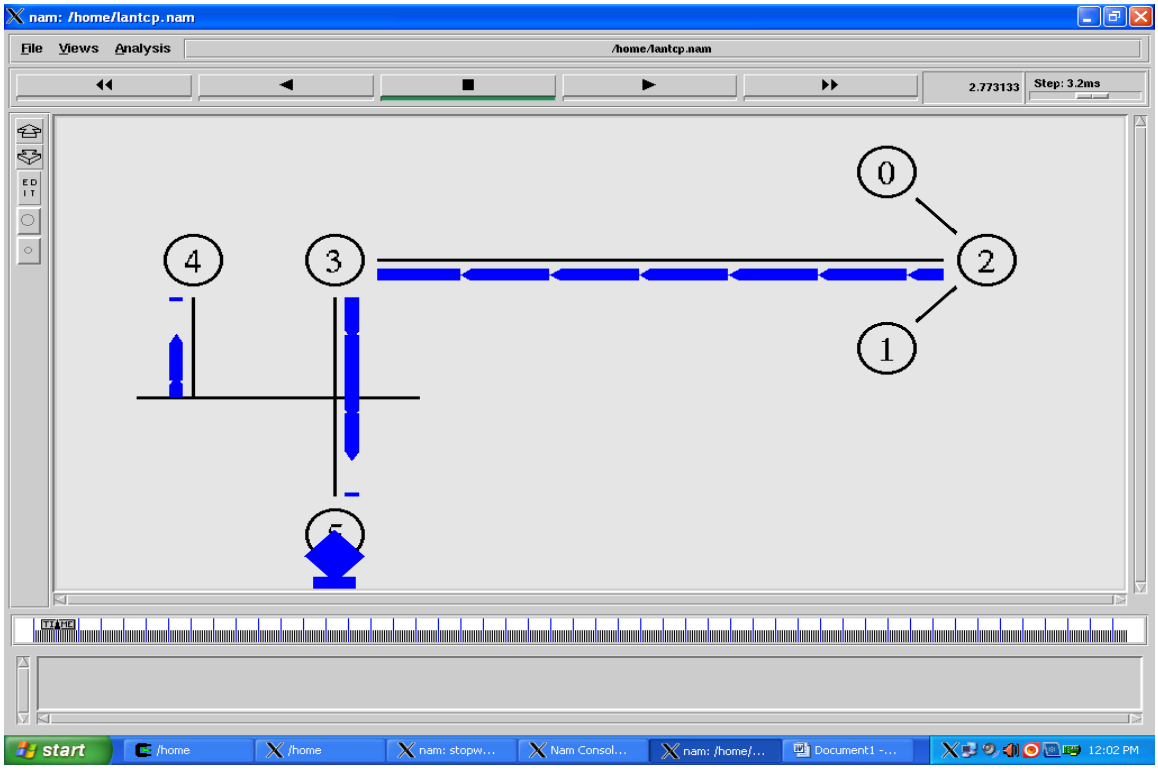
Step 1: Define different color for data flows for NAM
Step 2: Open the NAM trace file
Step 3: Define a finished procedure
Step 4: Define a desired Number of nodes
Step 5: Create a links between the nodes
Step 6: Specifies the position of the node for NAM
Step 7: Set the queue size between desired nodes
Step 8: Setup a TCP Connection
Step 9: Setup a FTP over TCP Connection
Step 10: Define a procedure for plotting window size
Step 11: Specify the end of simulation

PROGRAM FOR TCP PERFORMANCE

```
set ns [new Simulator]

$ns color 1 Blue
$ns color 2 Red

set tracefile1 [open lantcp.tr w]
set winfile [open WinFile w]
$ns trace-all $tracefile1
```

```
set namfile [open lantcp.nam w]
$ns namtrace-all $namfile

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$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 "$n4 $n5 $n3" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n3 $n2 orient left

$ns queue-limit $n2 $n3 20

set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set packetSize_ 552

set ftp [new Application/FTP]
$ftp attach-agent $tcp

$ns at 1.0 "$ftp  start"
$ns at 124.0 "$ftp  stop"

proc plotWindow {tcpSource file} {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotWindow $tcpSource $file"
}
$ns at 0.1 "plotWindow $tcp $winfile"
$ns run
```

OUTPUT:



Result:

To study the performance of UDP/TCP was done.

EX.NO.8.A SIMULATION OF DISTANCE VECTOR ALGORITHM.

Aim:

To write a ns2 program for implementing distance vector routing algorithm protocol.

Algorithm:

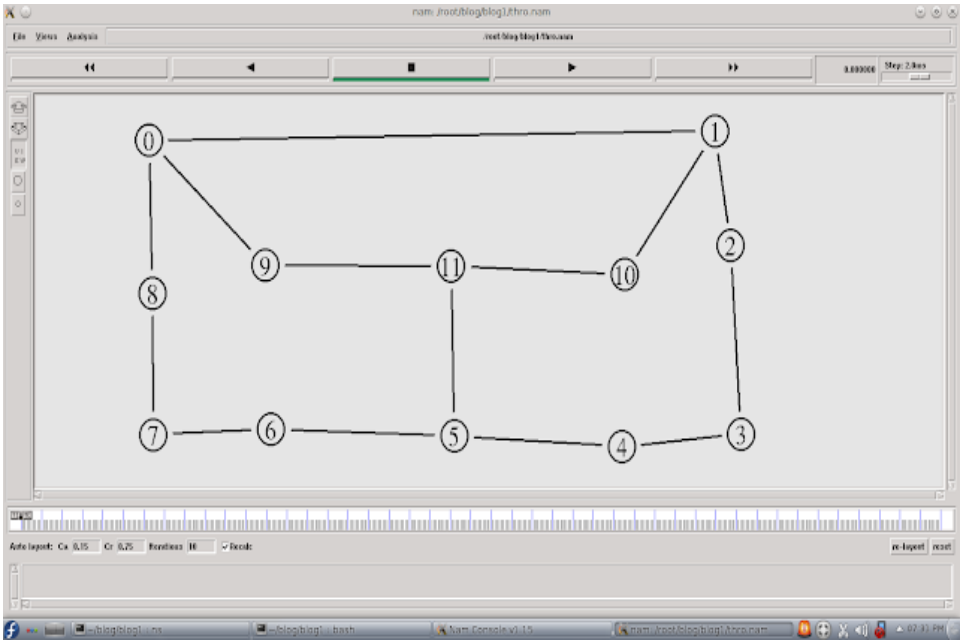
Step 1: start the program.
Step 2: declare the global variables ns for creating a new simulator.
Step 3: set the color for packets.
Step 4: open the network animator file in the name of file2 in the write mode.
Step 5: open the trace file in the name of file 1 in the write mode.
Step 6: set the unicast routing protocol to transfer the packets in network.
Step 7: create the required no of nodes.
Step 8: create the duplex-link between the nodes including the delay time,bandwidth and dropping . queue mechanism.
Step 9: give the position for the links between the nodes.
Step 10: set a tcp reno connection for source node.
Step 11: set the destination node using tcp sink.
Step 12: setup a ftp connection over the tcp connection.
Step 13: down the connection between any nodes at a particular time.
Step 14: reconnect the downed connection at a particular time.
Step 15: define the finish procedure.
Step 16: in the definition of the finish procedure declare the global variables ns,file1,file2.
Step 17: close the trace file and namefile and execute the network animation file.
Step 18: at the particular time call the finish procedure.
Step 19: stop the program.

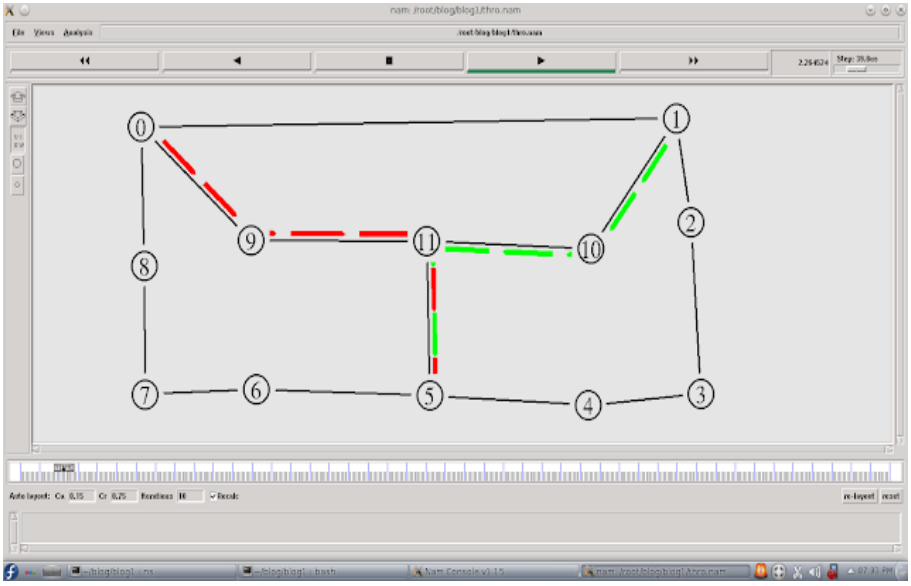
Program:

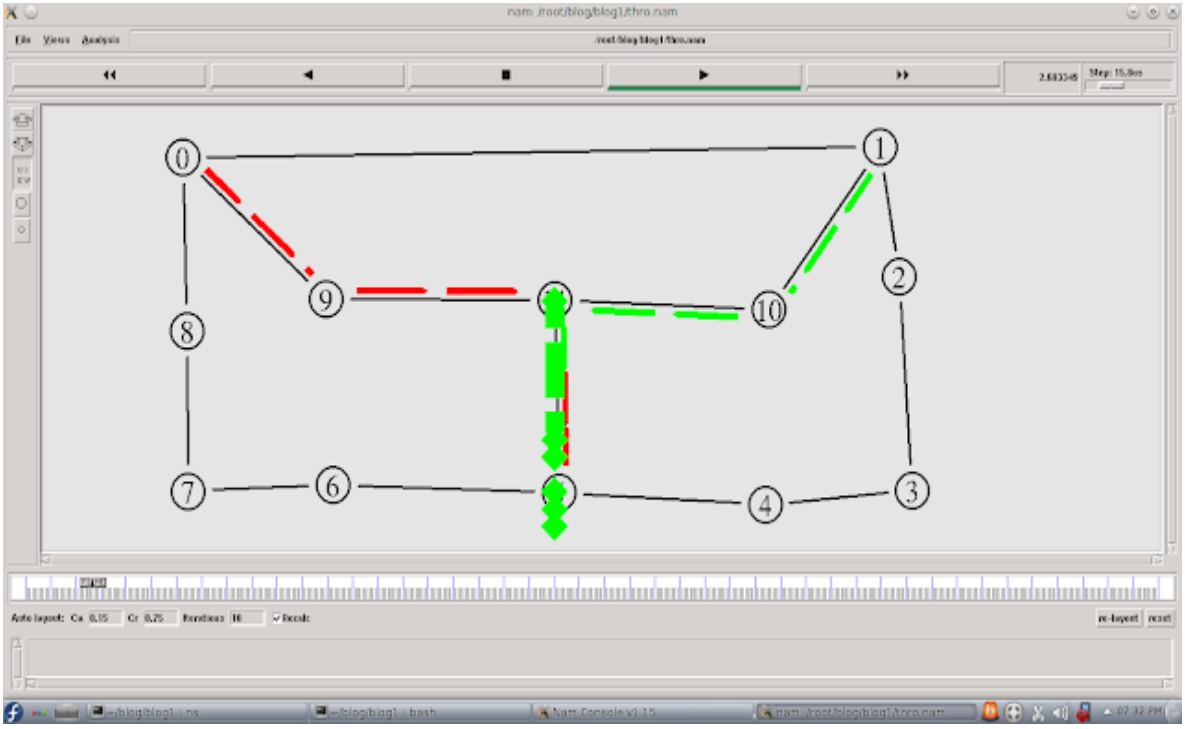
```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]
$ns namtrace-all $nf
      proc finish { } {
      global ns nr nf
      $ns flush-trace
      close $nf
      close $nr
      exec nam thro.nam &

exit 0
      }

for { set i 0 } { $i < 12} { incr i 1 } {
set n($i) [$ns node]}

for {set i 0} {$i < 8} {incr i} {
$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail }
```

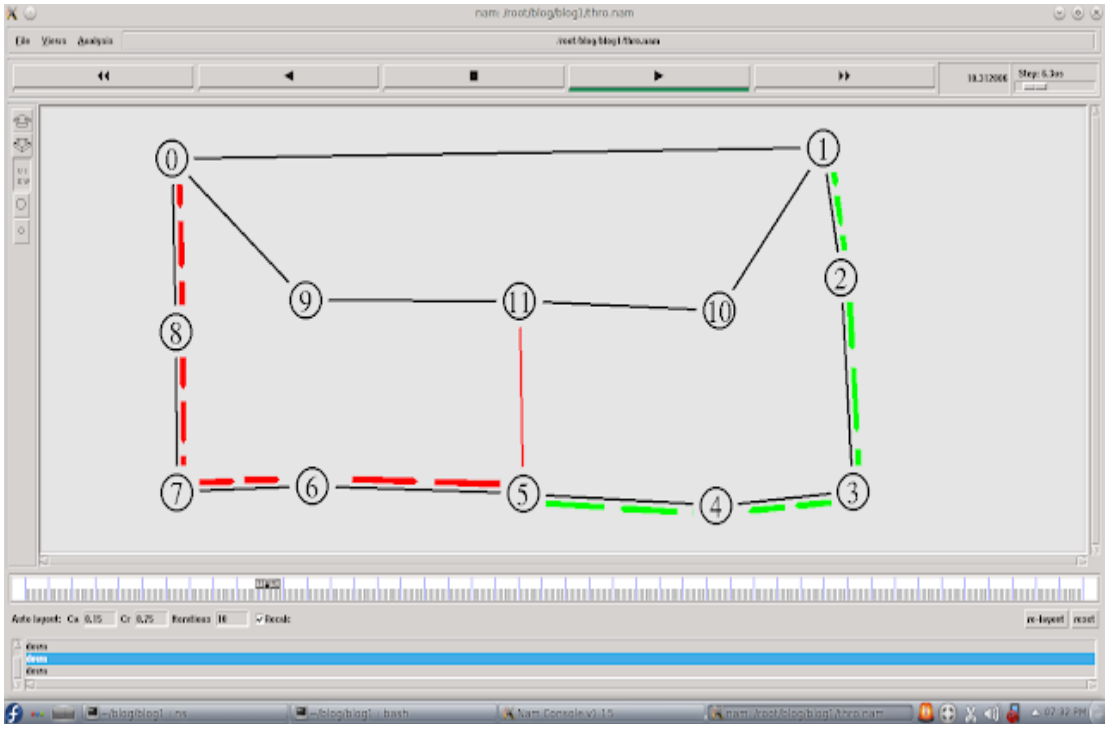
```
$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail

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(5) $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set  interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0

$ns rtproto DV
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)

$udp0 set fid_ 1
$udp1 set fid_ 2

$ns color 1 Red
$ns color 2 Green

$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"

$ns at 45 "finish"
$ns run
```

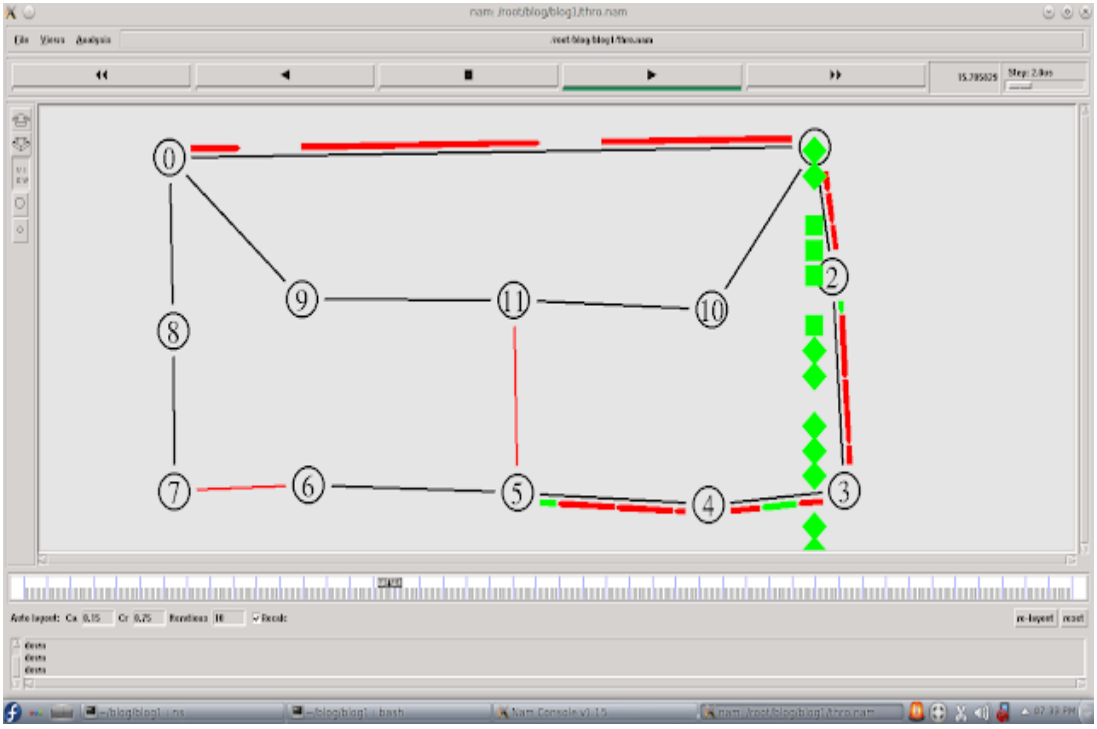
```
File  Edit  View  Bookmarks  Settings  Help
[root@deepu blog1]# ns dv.tcl
[root@deepu blog1]#
```






0
1
2
3
4
5
6
7
8
9
10


0
1
2
3
4
5
6
7
8
9
10
11



EX.NO.8(B) SIMULATION OF LINK STATE ROUTING ALGORITHM.

Aim:

To write a ns2 program for implementing of link state routing algorithm.

Step 1: start the program.
Step 2: declare the global variables ns for creating a new simulator.
Step 3: set the color for packets.
Step 4: open the network animator file in the name of file2 in the write mode.
Step 5: open the trace file in the name of file 1 in the write mode.
Step 6: set the multicast routing protocol to transfer the packets in network.
Step 7: create the multicast capable no of nodes.
Step 8: create the duplex-link between the nodes including the delay time, bandwidth and dropping . queue mechanism.
Step 9: give the position for the links between the nodes.
Step 10: set a udp connection for source node.
Step 11: set the destination node ,port and random false for the source and destination files.
Step 12: setup a traffic generator CBR for the source and destination files.
Step 13: down the connection between any nodes at a particular time.
Step 14: create the receive agent for joining and leaving if the nodes in the group.
Step 15: define the finish procedure.
Step 16: in the definition of the finish procedure declare the global variables.
Step 17: close the trace file and namefile and execute the network animation file.
Step 18: at the particular time call the finish procedure.
Step 19: stop the program.

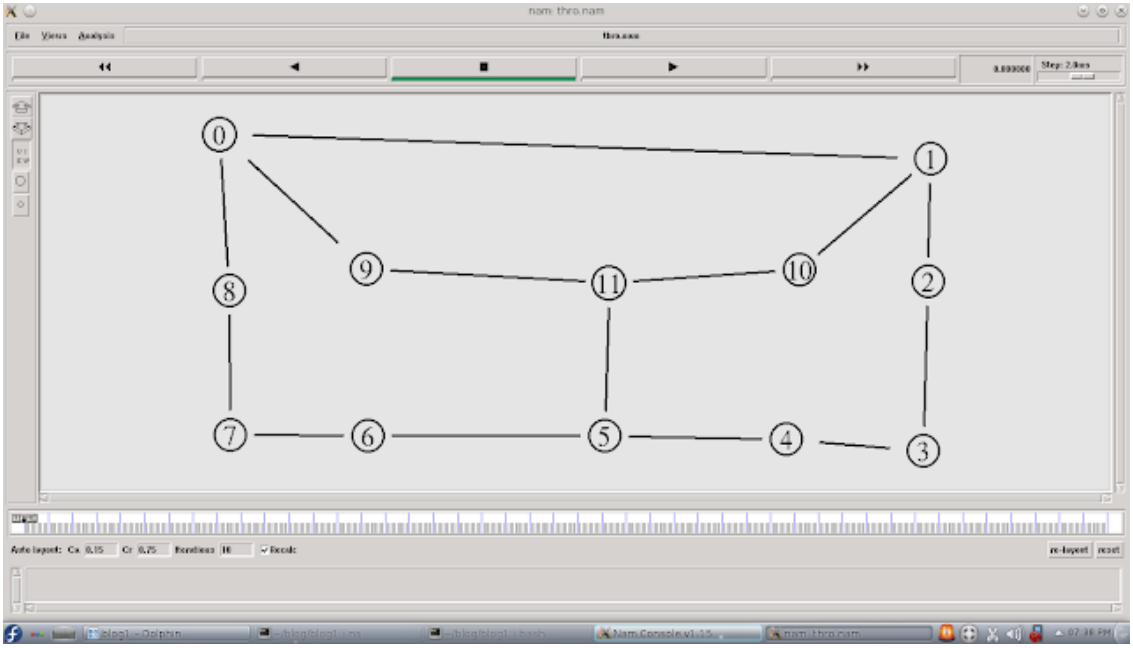
Program:

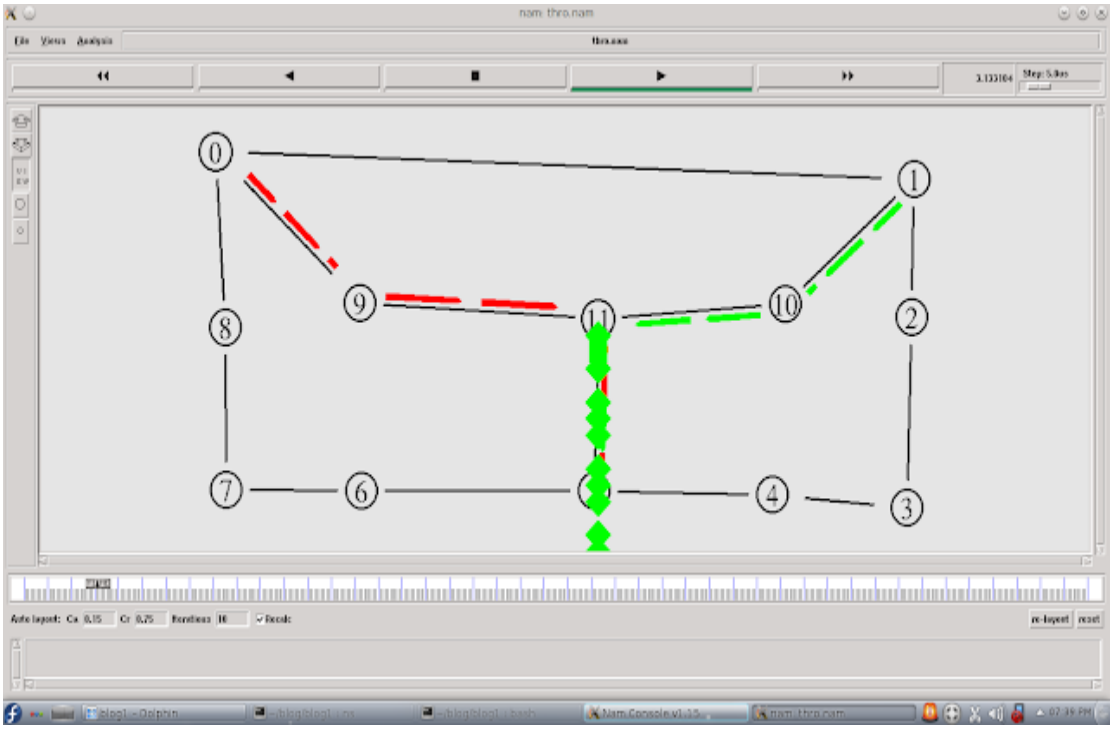
```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]

$ns namtrace-all $nf
      proc finish { } {
      global ns nr nf
      $ns flush-trace
      close $nf
      close $nr
      exec nam thro.nam &

exit 0
      }

for { set i 0 } { $i < 12} { incr i 1 } {
set n($i) [$ns node]}

for {set i 0} {$i < 8} {incr i} {
$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail  }

$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 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(5) $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set  interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
```

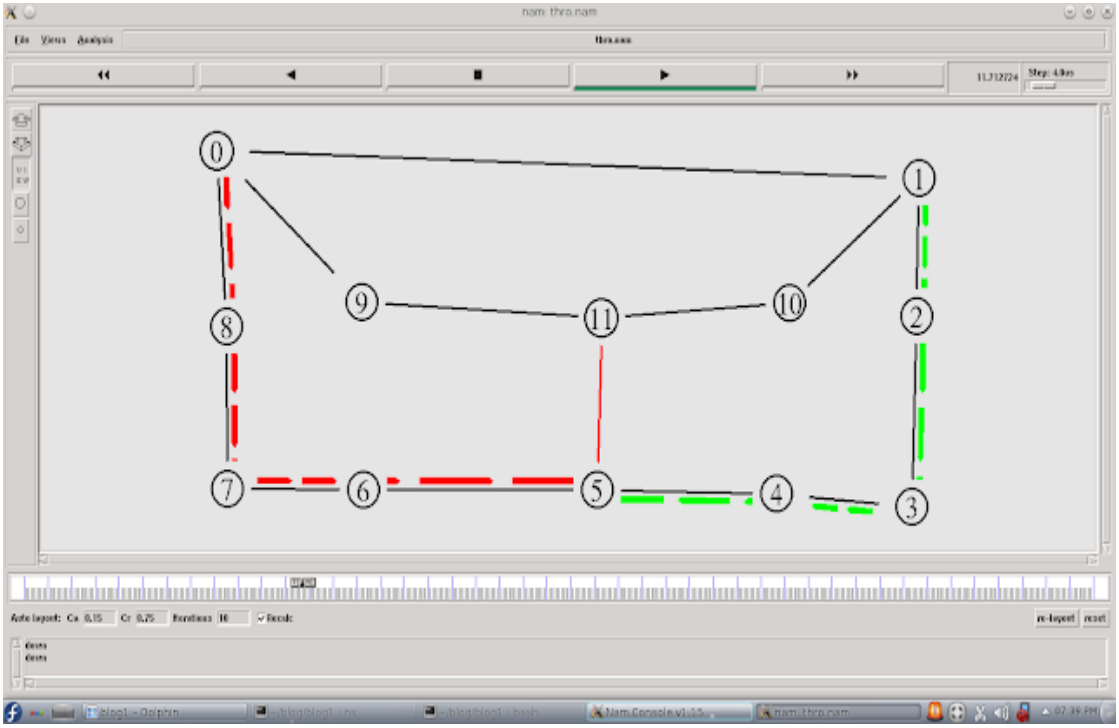
```
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0

$ns rtproto LS

$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)

$udp0 set fid_ 1
$udp1 set fid_ 2
$ns color 1 Red
$ns color 2 Green

$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"

$ns at 45 "finish"
$ns run
```

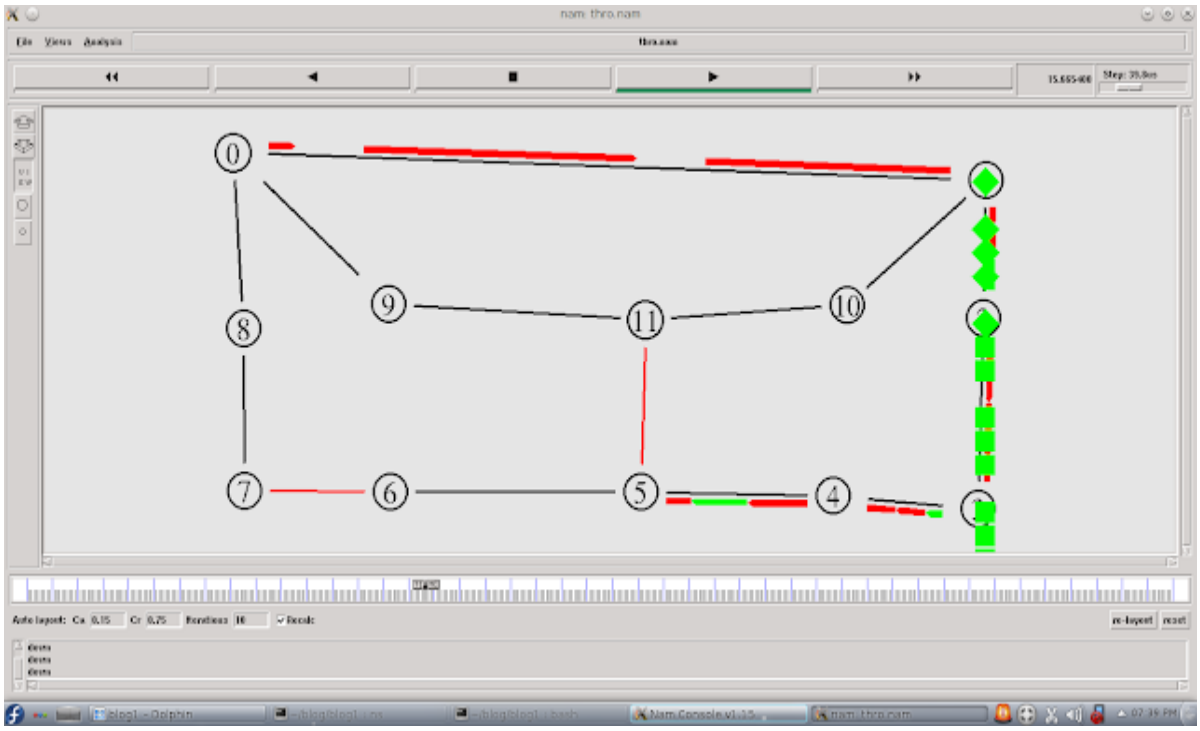
Output

```
File  Edit  View  Bookmarks  Settings  Help
[root@deepu blog1]# ns ls.tcl
[root@deepu blog1]#
```




nam thro.nam
0
1
2
3
4
5
6
7
8
9
10
11


nam thro.nam
0
1
2
3
4
5
6
7
8
9
10
11



Result:

Thus the ns2 program for implementing distance vector routing algorithm and link state routing algorithm was done.

EX NO: 9 PERFORMANCE COMPARISON OF AODV, DSDV and DSR ROUTING PROTOCOL

DATE:

AIM:

To write an NS2 program to compare the performance of AODV DSDV and DSR Routing protocols.

ALGORITHM:

Step 1: Start the program.

Step 2: Define the necessary definition for defining AODV, DSDV and DSR Routing protocol.

Step 3: Execute the program.

Step 4: Compare the throughput of the routing protocols by running the awk file each protocol for number of runs.

Step 5: Conclude the best routing protocol based on the result or by drawing the graph in excel by using throughput values.

Step 6: Stop the program.

PROGRAM

```
# Define options
set val(chan)         Channel/WirelessChannel    ;# channel type
set val(prop)         Propagation/TwoRayGround   ;# radio-propagation model
set val(netif)        Phy/WirelessPhy            ;# network interface type
set val(mac)          Mac/802_11                 ;# MAC type
set val(ifq)          Queue/DropTail/PriQueue    ;# interface queue type
set val(ll)           LL                         ;# link layer type
set val(ant)          Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)       50                         ;# max packet in ifq
set val(nn)           3                          ;# number of mobilenodes
set val(rp)           AODV                       ;# routing protocol
set val(x)            500                        ;# X dimension of topography
set val(y)            400                        ;# Y dimension of topography
set val(stop)         150                        ;# time of simulation end
set ns_               [new Simulator]
$ns_ use-newtrace
set tracefd       [open simple.tr w]
set windowVsTime2 [open win.tr w]
set namtrace      [open simple.nam w]
$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
# set up topography object
set topo       [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)
#
#  Create nn mobilenodes [$val(nn)] and attach them to the channel.
```

```
#
# configure the nodes
        $ns_ node-config -adhocRouting $val(rp) \
                         -llType $val(ll) \
                         -macType $val(mac) \
                         -ifqType $val(ifq) \
                         -ifqLen $val(ifqlen) \
                         -antType $val(ant) \
                         -propType $val(prop) \
                         -phyType $val(netif) \
                         -channelType $val(chan) \
                         -topoInstance $topo \
                         -agentTrace ON \
                         -routerTrace ON \
                         -macTrace ON \
                         -movementTrace ON

          for {set i 0} {$i < $val(nn) } { incr i } {
                  set node_($i) [$ns_ node]
          }
# Provide initial location of mobilenodes
$node_(0) set X_ 5.0
$node_(0) set Y_ 5.0
$node_(0) set Z_ 0.0
$node_(1) set X_ 490.0
$node_(1) set Y_ 285.0
$node_(1) set Z_ 0.0
$node_(2) set X_ 150.0
$node_(2) set Y_ 240.0
$node_(2) set Z_ 0.0
# Generation of movements
$ns_ at 10.0 "$node_(0) setdest 250.0 250.0 3.0"
$ns_ at 15.0 "$node_(1) setdest 45.0 285.0 5.0"
$ns_ at 110.0 "$node_(0) setdest 480.0 300.0 5.0"
# Set a TCP connection between node_(0) and node_(1)
set tcp [new Agent/TCP/Newreno]
$tcp set class_ 2
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(0) $tcp
$ns_ attach-agent $node_(1) $sink
$ns_ connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns_ at 10.0 "$ftp start"

# Printing the window size
proc plotWindow {tcpSource file} {
global ns_
```

```
set time 0.01
set now [$ns_ now]
set cwnd [$tcpSource set cwnd_]
puts $file "$now $cwnd"
$ns_ at [expr $now+$time] "plotWindow $tcpSource $file" }
$ns_ at 10.1 "plotWindow $tcp $windowVsTime2"

# Define node initial position in nam
for {set i 0} {$i < $val(nn)} { incr i } {
# 30 defines the node size for nam
$ns_ initial_node_pos $node_($i) 30
}
# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
   $ns_ at $val(stop) "$node_($i) reset";
}
# ending nam and the simulation
$ns_ at $val(stop) "$ns_ nam-end-wireless $val(stop)"
$ns_ at $val(stop) "stop"
$ns_ at 150.01 "puts \"end simulation\" ; $ns_ halt"
proc stop {} {
   global ns_ tracefd namtrace
   $ns_ flush-trace
   close $tracefd
   close $namtrace
}
```

PDR.awk

```
BEGIN {
      recvdSize = 0
      startTime = 400
      stopTime = 0
  }
    {
           event = $1
           time = $3
           node_id = $41
           pkt_size = $37
           level = $19
    # Store start time
  if (level == "AGT" && event == "s" && pkt_size >= 512) {
    if (time < startTime) {
           startTime = time
           }
      }
    # Update total received packets' size and store packets arrival time
  if (level == "AGT" && event == "r" && pkt_size >= 512) {
      if (time > stopTime) {
           stopTime = time
```

```
        }
    # Rip off the header
    hdr_size = pkt_size % 512
    pkt_size -= hdr_size
    # Store received packet's size
    recvdSize += pkt_size
    }
  }
    END {
    printf("Average Throughput[kbps] = %.2f\t\t StartTime=%.2f\tStopTime=%.2f\n",(recvdSize/(stopTime-
startTime))*(8/1000),startTime,stopTime)
  }
$ns_ run
```

SAMPLE OUTPUT:

AODV:
Average Throughput[kbps]: 340
Average Throughput[kbps]: 320
Average Throughput[kbps]: 360
DSDV:
Average Throughput[kbps]: 320
Average Throughput[kbps]: 310
Average Throughput[kbps]: 300
DSR:
Average Throughput[kbps]: 280
Average Throughput[kbps]: 270
Average Throughput[kbps]: 275

Result:

Thus the performance comparison of AODV, DSDV and DSR routing protocol was evauated.

EX.NO:10 IMPLEMENTATION OF CRC USING RAW SOCKETS

DATE:

AIM:

To write a program to establish connection between client and server using CRC

ALGORITHM:

CRC SERVER

Step 1: Start the program.
Step 2: Create an object for CRC 32.
Step 3: Declare s and create object for server socket.
Step 4: For string tokenizer create an object st.
Step 5: To update message use update command.
Step 6: If s1 and s2 are equal then print a message.
Step 7: Catch the exception
Step 8 : stop the program

CRC CLIENT:-

Step 1: Start the program.
Step 2: Create an object for Inet Address and CRC 32, buffer.
Step 3: Get the lang value for server.
Step 4: Create object for DataInputStream.
Step 5: Read the message line using str object.
Step 6: Now acknowledge will be received from object.
Step 7: Catch the exception
Step 8: Stop the program.

PROGRAM

//crcserver.java

```
import java.io.*;
import java.net.*;
import java.util.zip.*;
import java.util.*;
class crcserver
{
public static void main(String ar[])
{
try
{
CRC32 c= new CRC32();
ServerSocket ss= new ServerSocket(8000);
Socket s;
String str;
String sis[]=new String[500];
while(true)
{
s=ss.accept();
```

```
DataInputStream dis=new DataInputStream(s.getInputStream());
str=dis.readLine();
System.out.println("The message with the CRC value received is \n");
System.out.println(str);
StringTokenizer st= new StringTokenizer(str,"//");
int n=st.countTokens();
System.out.println("Number of token:"+n);
System.out.println("The token received are:");
for(int i=0;i<n;i++)
{
sis[i]=st.nextToken();
System.out.println(sis[i]);
}
long val1=Long.parseLong(sis[n-1]);
String s1=Long.toString(val1);
System.out.println("The calculate crc value is\n");
System.out.println(val1);
c.update(sis[0].getBytes());
long val=c.getValue();
String s2=Long.toString(val);
if(s1.equals(s2))
{
String s3="Good Messsage";
PrintStream ps=new PrintStream(s.getOutputStream());
ps.println(s3);
}
System.out.println("The message from Client is accepted");
}
}
catch(Exception e)
{
System.out.println(e);}}}
```

//crcclient.java

```
import java.net.*;
import java.io.*;
import java.util.zip.*;
class crcclient
{
public static void main(String ar[])
{
try
{
InetAddress ia=InetAddress.getLocalHost();
CRC32 cc=new CRC32();
System.out.println("Please enter a message");
BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
String s=br.readLine();
```

```
PrintStream ps;
String m;
Socket ss= new Socket(ia,8000);
cc.update(s.getBytes());
long val= cc.getValue();
ps=new PrintStream(ss.getOutputStream());
m=s+"//"+val;
System.out.println("The message and crc value is \n");
System.out.println(m);
ps.println(m);
DataInputStream dis= new DataInputStream(ss.getInputStream());
String str=dis.readLine();
System.out.println("The Acknowlegement received from serveris \n");
System.out.println(str);
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

OUTPUT:

```
> javac crcclient.java
> java crcclient

        Please enter a message
        >Welcome
        The message and crc value is
        Welcome // 936075699
        The acknowledgement received from server is
        Good Message

> javac crcserver.java
> java crcserver

        The messasge with crc value received is
        Welcome //936075699
        Number of token : 2
        The token received are:
        Welcome
        936075699
        The message from client is accepted
```

Result:

Thus the Simulation of error correction code was done.