

<b>EX NO: 1.A</b>	Write a code simulating PING command
<b>DATE:</b>	

**Aim:**

To write a java program for simulating ping command.

**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**

### **//pingclient.java**

```
import java.io.*;
import java.net.*;
import java.util.Calendar;
class pingclient
{
    public static void main(String args[])throws Exception
    {
        String str;
        int c=0;
        long t1,t2;
        Socket s=new Socket("127.0.0.1",5555);
        DataInputStream dis=new DataInputStream(s.getInputStream());
        PrintStream out=new PrintStream(s.getOutputStream());
        while(c<4)
        {
            t1=System.currentTimeMillis();
            str="Welcome to network programming world";
            out.println(str);
            System.out.println(dis.readLine());
            t2=System.currentTimeMillis();
            System.out.println("TTL="+t2-t1+"ms");
            c++;
        }
        s.close();
    }
}
```

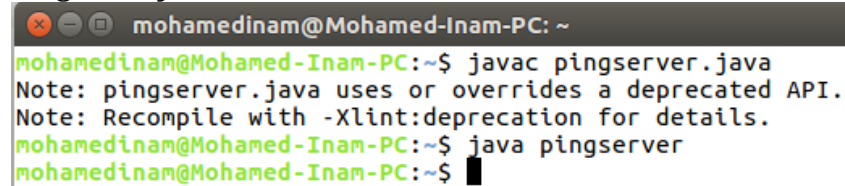
### **//pingserver.java**

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.text.*;
class pingserver
{
    public static void main(String args[])throws Exception
    {
        ServerSocket ss=new ServerSocket(5555);
        Socket s=ss.accept();
        int c=0;
        while(c<4)
        {
            DataInputStream dis=new DataInputStream(s.getInputStream());
            PrintStream out=new PrintStream(s.getOutputStream());
```

```
String str=dis.readLine();
out.println("Reply from"+InetAddress.getLocalHost()
+";Length"+str.length());
c++;
}
s.close();
} }
```

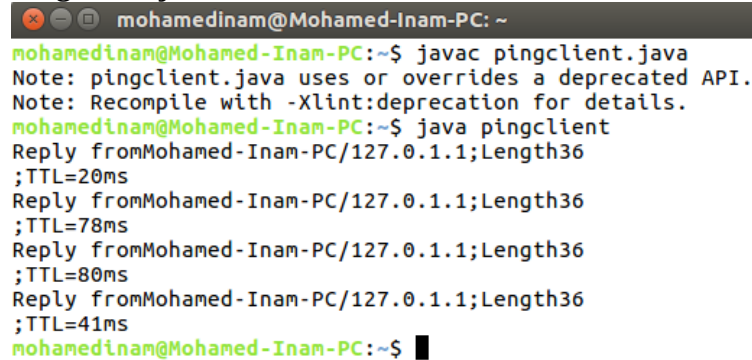
## OUTPUT:

### Pingserver.java

A terminal window titled 'mohamedinam@Mohamed-Inam-PC: ~' showing the compilation and execution of 'pingserver.java'. The commands and their outputs are: 'javac pingserver.java' with two deprecation warnings, and 'java pingserver' which runs successfully without output.

```
mohamedinam@Mohamed-Inam-PC: ~$ javac pingserver.java
Note: pingserver.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
mohamedinam@Mohamed-Inam-PC: ~$ java pingserver
mohamedinam@Mohamed-Inam-PC: ~$
```

### PingClient.java

A terminal window titled 'mohamedinam@Mohamed-Inam-PC: ~' showing the compilation and execution of 'pingclient.java'. The commands and their outputs are: 'javac pingclient.java' with two deprecation warnings, and 'java pingclient' which outputs five ping results with TTL values and lengths.

```
mohamedinam@Mohamed-Inam-PC: ~$ javac pingclient.java
Note: pingclient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
mohamedinam@Mohamed-Inam-PC: ~$ java pingclient
Reply fromMohamed-Inam-PC/127.0.1.1;Length36
;TTL=20ms
Reply fromMohamed-Inam-PC/127.0.1.1;Length36
;TTL=78ms
Reply fromMohamed-Inam-PC/127.0.1.1;Length36
;TTL=80ms
Reply fromMohamed-Inam-PC/127.0.1.1;Length36
;TTL=41ms
mohamedinam@Mohamed-Inam-PC: ~$
```

## Result:

Thus the implementation of ping command is executed successfully.

<b>EX NO: 1b</b>	Write a code simulating TRACEROUTE command
<b>DATE:</b>	

**Aim:**

To write a java program for simulating traceroute command.

**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.

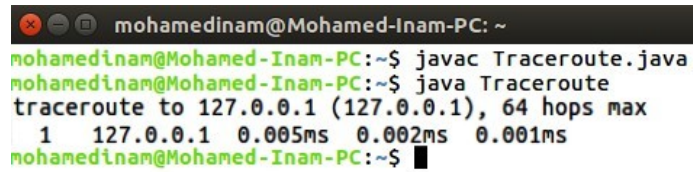
**Program:**

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

class Traceroute
{
    public static void main(String args[]){
        BufferedReader in;
        try{
            Runtime r = Runtime.getRuntime();
            Process p = r.exec("tracert 127.0.0.1");
            in = new BufferedReader(new
InputStreamReader(p.getInputStream()));
            String line;
            if(p==null)
                System.out.println("could not connect");
            while((line=in.readLine())!=null){
                System.out.println(line);
                in.close();
            }
        }catch(IOException e){
            System.out.println(e.toString());
        }
    }
}
```

## OUTPUT:

### Traceroute:



```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac Traceroute.java  
mohamedinam@Mohamed-Inam-PC:~$ java Traceroute  
traceroute to 127.0.0.1 (127.0.0.1), 64 hops max  
 1  127.0.0.1  0.005ms  0.002ms  0.001ms  
mohamedinam@Mohamed-Inam-PC:~$
```

### Result:

Thus the implementation of trace route command is executed and verified successfully.

EX NO: 2	<b>CREATE A SOCKET FOR HTTP WEBPAGE UPLOAD AND DOWNLOAD</b>
DATE:	

**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:

### Server.java

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac Server.java  
mohamedinam@Mohamed-Inam-PC:~$ java Server  
Server Waiting for image  
Client connected.  
Image Size: 405KB  
█
```

### Client.java

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac Client.java  
mohamedinam@Mohamed-Inam-PC:~$ java Client  
Client is running.  
Reading image from disk.  
Sending image to server.  
Image sent to server.  
mohamedinam@Mohamed-Inam-PC:~$ █
```

### Server:



## Result:

Thus the implementation of HTTP for webpage upload and download is executed successfully.

EX NO: 3a	<b>IMPLEMENTATION OF APPLICATION USING SOCKET (ECHO CLIENT AND SERVER)</b>
DATE:	

**AIM**

To write a java program for Echo client and echo server using TCP Sockets.

**ALGORITHM**

1. Create two programs one for the server side and one for the client side
2. In the server side, create a server socket.
3. The return value of the accept () method is assigned to a new socket created.
4. Send the input received from the client at the server side back to the client.
5. In the client side, create a socket to connect to the server.
6. Create the object of DataInputStream to accept input from the server
7. Display the input received from the server.
8. 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();
            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:

### Echoserver

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac EServer.java  
Note: EServer.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
mohamedinam@Mohamed-Inam-PC:~$ java EServer  
ds  
█
```

### EchoClient

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac EClient.java  
Note: EClient.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
mohamedinam@Mohamed-Inam-PC:~$ java EClient  
Client:  
Hi Server  
Server:Hi Server  
Client:  
How r u  
Server:How r u  
Client:  
fyn here  
Server:fyn here  
Client:  
ds  
Server:ds  
Client:  
ds  
Socket Closed!█
```

## Result:

Thus the implementation of TCP socket application i.e. Echo server and client is executed and verified successfully.

EX NO: 3b	<b>IMPLEMENTATION OF APPLICATION USING SOCKET (CLIENT SERVER SYSTEM- CHAT APPLICATION)</b>
DATE:	

**AIM**

To write a java program for chat using TCP Sockets.

**ALGORITHM****Server**

1. Start the program
2. Create server and client sockets.
3. Use input streams to get the message from user.
4. Use output streams to send message to the client.
5. Wait for client to display this message and write a new one to be displayed by the server.
6. Display message given at client using input streams read from socket.
7. Stop the program.

**Client**

1. Start the program
2. Create a client socket that connects to the required host and port.
3. Use input streams read message given by server and print it.
4. Use input streams; get the message from user to be given to the server.
5. Use output streams to write message to the server.
6. Stop the program.

## **PROGRAM:**

### **chatserver.java**

```
import java.io.*;
import java.net.*;
class chatserver
{
    public static DatagramSocket ds;
    public static byte buffer[]=new byte[1024];
    public static int clientport=7890,serverport=7900;
    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.getLocalHost();
        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));
        }
    }
}
```

### **chatclient.java**

```
import java .io.*;
import java.net.*;
class chatclient
{
    public static DatagramSocket ds;
    public static int clientport=7890,serverport=7900;
    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:

### ChatServer:

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac UDPserver.java  
mohamedinam@Mohamed-Inam-PC:~$ java UDPserver  
press ctrl+c to quit the program  
Client:hi client  
Server:  
hi server  
Client:how r u  
Server:  
im fyn wht abt u  
Client:fyn  
Server:  
bye  
Client:bye  
Server:  
█
```

### Chat client

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac UDPclient.java  
mohamedinam@Mohamed-Inam-PC:~$ java UDPclient  
server waiting  
Client:  
hi client  
Server:hi server  
Client:  
how r u  
Server:im fyn  
Client:  
fyn  
bye  
Server:bye  
Client:  
█
```

## Result:

Thus implementation of chat system using socket programming is executed and verified successfully.

EX NO: 3c	<b>IMPLEMENTATION OF APPLICATION USING SOCKET (FILE TRANSFER)</b>
DATE:	

**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 BufferedReader class is used for storing data content which has been retrieved from socket object.
5. The content of file is displayed in the client window and the connection is closed.
6. Stop the program.

## **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",1390);
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);
}
}
}
```

### **File 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:

### FileServer:

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac Serverfile.java  
Note: Serverfile.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
mohamedinam@Mohamed-Inam-PC:~$ java Serverfile  
MY FILE  
ajnlaksdfb  
sfjsklafsbh  
sfkjasdhfiu  
jgkfnkinam  
inogijoam  
█
```

### FileClient:

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac Clientfile.java  
Note: Clientfile.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
mohamedinam@Mohamed-Inam-PC:~$ java Clientfile  
Enter the file name:  
temp.txt  
Enter the new file name:  
new.txt  
MY FILE  
ajnlaksdfb  
sfjsklafsbh  
sfkjasdhfiu  
jgkfnkinam  
inogijoam  
mohamedinam@Mohamed-Inam-PC:~$ █
```

## Result:

Thus the implementation of File transfer using TCP socket is executed and verified successfully.

EX NO: 4	<b>IMPLEMENTATION OF APPLICATION USING UDP (DNS)</b>
DATE:	

**AIM:**

To write a java program for DNS application.

**ALGORITHM****Server**

- Step1: Start the program.
- Step2: Create the socket for the server.
- Step3: Bind the socket to the port.
- Step4: Listen for the incoming client connection.
- Step5: Receive the IP address from the client to be resolved.
- Step6: Get the domain name for the client.
- Step7: Check the existence of the domain in the server.
- Step8: If domain matches then send the corresponding address to the client.
- Step9: Stop the program execution

**Client**

- Step1: Start the Program.
- Step2: Create the socket for the client.
- Step3: Connect the socket to the Server.
- Step4: Send the host name to the server to be resolved.
- Step5: If the server corresponds then print the address and terminate the process

**PROGRAM:****UDP DNS Server****Udpdnsserver**

```
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();
}
}

```

## OUTPUT

### DNSserver:

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac udpdnsserver.java  
mohamedinam@Mohamed-Inam-PC:~$ java udpdnsserver  
Press Ctrl + C to Quit  
Request for host yahoo.com  
Request for host facebook.com  
Request for host duckduckgo.com  
█
```

### DNSclient:

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac udpdnsclient.java  
mohamedinam@Mohamed-Inam-PC:~$ java udpdnsclient  
Enter the hostname : yahoo.com  
IP Address: 68.180.206.184  
mohamedinam@Mohamed-Inam-PC:~$ java udpdnsclient  
Enter the hostname : facebook.com  
IP Address: 69.63.189.16  
mohamedinam@Mohamed-Inam-PC:~$ java udpdnsclient  
Enter the hostname : duckduckgo.com  
IP Address: Host Not Found  
mohamedinam@Mohamed-Inam-PC:~$ █
```

### Result:

Thus the implementation of DNS server and client using UDP socket is executed successfully.

EX NO: 5 a	<b>WRITE A CODE FOR SIMULATING ARP PROTOCOL</b>
DATE:	

**Aim:**

To write a java program for simulating Address Resolution Protocol(6) 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",1390);
            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.*;
c lass 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:

### Server:

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac Serverarp.java  
Note: Serverarp.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
mohamedinam@Mohamed-Inam-PC:~$ java Serverarp  
  
java.lang.NullPointerException  
mohamedinam@Mohamed-Inam-PC:~$  
mohamedinam@Mohamed-Inam-PC:~$
```

### Client:

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac Clientarp.java  
Note: Clientarp.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
mohamedinam@Mohamed-Inam-PC:~$ java Clientarp  
Enter the Logical address(IP):  
165.165.80.80  
The Physical Address is: 6A:08:AA:C2  
mohamedinam@Mohamed-Inam-PC:~$
```

### Result:

Thus the implementation of ARP(Address Resolution Protocol) is implemented successfully.

EX NO: 5b	<b>WRITE A CODE FOR SIMULATING RARP PROTOCOL</b>
DATE:	

**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.

## **PROGRAM:**

### **Client:**

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientarp12
{
    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

### Serverrrarp:

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac Serverrrarp12.java  
mohamedinam@Mohamed-Inam-PC:~$ java Serverrrarp12  
mohamedinam@Mohamed-Inam-PC:~$ █
```

### Clientrrarp

```
mohamedinam@Mohamed-Inam-PC: ~  
mohamedinam@Mohamed-Inam-PC:~$ javac Clientrrarp12.java  
mohamedinam@Mohamed-Inam-PC:~$ java Clientrrarp12  
Enter the Physical address (MAC):  
8A:BC:E3:FA  
The Logical Address is(IP): 165.165.79.1  
mohamedinam@Mohamed-Inam-PC:~$ █
```

### Result:

Thus the implementation of RARP (Reverse Address Resolution Protocol) is implemented successfully.

EX NO: 6

DATE:

## STUDY OF NETWORK SIMULATOR (NS) AND SIMULATION OF CONGESTION CONTROL ALGORITHMS USING NS

### Aim:

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

### NET WORK SIMULATOR (NS2)

#### Ns overview

- Ns programming: A Quick start
- Case study I: A simple Wireless network
- Case study II: Create a new agent in NS

#### Ns overview

- Ns Status
- Periodical release (ns2)
- Platform support
- FreeBSD, Linux, Solaris, Windows and Mac

#### Ns functionalities

Routing, Transportation, Traffic sources, Queuing disciplines, QoS

#### Wireless

Ad hoc routing, mobile IP, sensor-MAC  
Tracing, visualization and various utilities  
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, analyse 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** 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 transport layer 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**

This 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. The throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second or data packets per time slot. This measure how soon the receiver is able to get a certain amount of data sent 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 egress. The larger the value of delay, the more difficult it is for transport layer protocols to maintain high bandwidths. 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.

### **RESULT:**

Thus the study of network simulators (NS) and simulation of congestion control algorithm using NS is studied successfully.

EX NO: 7	CASE STUDY ON VARIOUS ROUTING ALGORITHM
DATE:	

### AIM

To study the various routing algorithm.

### 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.

## ii. 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 neighbors and route packets indefinitely. More than two neighbors create 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.

### Algorithm

There are several variants of flooding algorithm. Most work roughly as follows:

1. Each node acts as both a transmitter and a receiver.
2. Each node tries to forward every message to every one of its neighbors except the source node.

This results in every message eventually being delivered to all reachable parts of the network. Algorithms may need to be more complex than this, since, in some case, precautions have to be taken to avoid wasted duplicate deliveries and infinite loops, and to allow messages to eventually expire from the system. A variant of flooding called *selective flooding* partially addresses these issues by only sending packets to routers in the same direction. In selective flooding the routers don't send every incoming packet on every line but only on those lines which are going approximately in the right direction.

### Advantages

- If a packet can be delivered, it will (probably multiple times).
- Since flooding naturally utilizes every path through the network, it will also use the shortest path.
- This algorithm is very simple to implement.

### **Disadvantages**

- Flooding can be costly in terms of wasted bandwidth. While a message may only have one destination it has to be sent to every host. In the case of a ping flood or a denial of service attack, it can be harmful to the reliability of a computer network.
- Messages can become duplicated in the network further increasing the load on the networks bandwidth as well as requiring an increase in processing complexity to disregard duplicate messages.
- Duplicate packets may circulate forever, unless certain precautions are taken:
- Use a hop count or a time to live count and include it with each packet. This value should take into account the number of nodes that a packet may have to pass through on the way to its destination.
- Have each node keep track of every packet seen and only forward each packet once
- Enforce a network topology without loops.

### **iii . Distance vector**

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' 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).

### **RESULT**

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

<b>EX NO: 8</b>	<b>DISTANCE VECTOR ROUTING</b>
<b>DATE:</b>	

**AIM:**

To write a ns2 program for implementing unicast routing 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:**

```
//TCP//
set ns [new Simulator]

set nf [open outdvp.nam w]
$ns namtrace-all $nf

set tr [open outdvp.tr w]
$ns trace-all $tr

proc finish {} {
    global nf ns tr
    $ns flush-trace
    close $tr
    exec nam outdvp.nam &
    exit 0
}

#Define different colors for data flows
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green

#Create seven nodes
for {set i 0} {$i < 10} {incr i} {
    set n($i) [$ns node]
}

#Create links between the nodes
for {set i 0} {$i < 10} {incr i} {
    $ns duplex-link $n($i) $n([expr ($i+1)%10]) 1Mb 10ms DropTail
}

$ns duplex-link $n(0) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(8) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(7) $n(5) 1Mb 10ms DropTail
$ns duplex-link $n(2) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(4) $n(8) 1Mb 10ms DropTail

$ns rtproto DV

set tcp0 [new Agent/TCP]
```

```
$ns attach-agent $n(0) $tcp0
```

```
set ftp0 [new Application/FTP]  
$ftp0 attach-agent $tcp0
```

```
#$tcp set class_ 1
```

```
set sink0 [new Agent/TCPSink]  
$ns attach-agent $n(5) $sink0
```

```
$ns connect $tcp0 $sink0
```

```
set tcp1 [new Agent/TCP]  
$ns attach-agent $n(3) $tcp1
```

```
set ftp1 [new Application/FTP]  
$ftp1 attach-agent $tcp1
```

```
#$tcp set class_ 1
```

```
set sink1 [new Agent/TCPSink]  
$ns attach-agent $n(7) $sink1
```

```
$ns connect $tcp1 $sink1
```

```
set tcp2 [new Agent/TCP]  
$ns attach-agent $n(4) $tcp2
```

```
set ftp2 [new Application/FTP]  
$ftp2 attach-agent $tcp2
```

```
$tcp2 set class_ 1
```

```
set sink2 [new Agent/TCPSink]  
$ns attach-agent $n(9) $sink2
```

```
$ns connect $tcp2 $sink2
```

```
$tcp0 set fid_ 1  
$tcp1 set fid_ 2  
$tcp2 set fid_ 3
```

```
$ns at 0.5 "$ftp0 start"  
$ns at 1.5 "$ftp1 start"  
$ns at 2.5 "$ftp2 start"
```

```
$ns at 3.5 "$ftp0 stop"
$ns at 4.5 "$ftp1 stop"
$ns at 5.9 "$ftp2 stop"
$ns at 6.0 "finish"
```

```
$ns run
```

#### **//TCP AWK FILE**

```
BEGIN {
    recvdSize = 0
    startTime = 0
    stopTime = 0
}

{

    # Trace line format: new
    # if ($2 == "-t") {
        event = $1
        time = $2
        pkt_id = $12
        pkt_size = $6
        level = $5
    #}

    # Store start time
    if (level == "tcp" && (event == "+" || event == "s") && pkt_size >= 500)
    {
        if (time < startTime) {
            startTime = time
        }
    }

    # Update total received packets' size and store packets arrival time
    if (level == "tcp" && event == "r" && pkt_size >= 500) {
        if (time > stopTime) {
            stopTime = time
        }
        # Rip off the header
        hdr_size = pkt_size % 500
        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)
}

//UDP
set ns [new Simulator]

set nf [open outdvp.nam w]
$ns namtrace-all $nf

set tr [open outdvp.tr w]
$ns trace-all $tr

proc finish {} {
    global nf ns tr
    $ns flush-trace
    close $tr
    exec nam outlsp.nam &
    exit 0
}

#Define different colors for data flows
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green

#Create seven nodes
for {set i 0} {$i < 10} {incr i} {
    set n($i) [$ns node]
}

#Create links between the nodes
for {set i 0} {$i < 10} {incr i} {
    $ns duplex-link $n($i) $n([expr ($i+1)%10]) 1Mb 10ms DropTail
}

$ns duplex-link $n(0) $n(3) 1Mb 10ms DropTail

```

```
$ns duplex-link $n(8) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(7) $n(5) 1Mb 10ms DropTail
$ns duplex-link $n(2) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(4) $n(8) 1Mb 10ms DropTail
```

```
$ns rtproto DV
```

```
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
$udp0 set class_ 0
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(4) $udp1
$udp1 set class_ 1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null1 [new Agent/Null]
$ns attach-agent $n(5) $null1
$ns connect $udp1 $null1
```

```
set udp2 [new Agent/UDP]
$ns attach-agent $n(3) $udp2
$udp2 set class_ 2
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 500
$cbr2 set interval_ 0.005
$cbr2 attach-agent $udp2
set null2 [new Agent/Null]
$ns attach-agent $n(7) $null2
$ns connect $udp2 $null2
```

```
$udp0 set fid_ 1
$udp1 set fid_ 2
$udp2 set fid_ 3
```

```
#Schedule events for the CBR agent and the network dynamics
```

```
$ns at 0.5 "$cbr0 start"
$ns at 1.5 "$cbr1 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 2.5 "$cbr2 start"
$ns at 3.5 "$cbr0 stop"
$ns at 4.5 "$cbr1 stop"
$ns at 5.9 "$cbr2 stop"
$ns at 6.0 "finish"
```

```
$ns run
```

```
//UDP AWK FILE//
```

```
BEGIN {
    recvdSize = 0
    startTime = 0
    stopTime = 0
}

{

    # Trace line format: new
    # if ($2 == "-t") {
        event = $1
        time = $2
        pkt_id = $12
        pkt_size = $6
        level = $5
    #}

    # Store start time
    if (level == "cbr" && (event == "+" || event == "s") && pkt_size >= 500)
    {
        if (time < startTime) {
            startTime = time
        }
    }

    # Update total received packets' size and store packets arrival time
    if (level == "cbr" && event == "r" && pkt_size >= 500) {
        if (time > stopTime) {
            stopTime = time
        }
        # Rip off the header
        hdr_size = pkt_size % 500
        pkt_size -= hdr_size
    }
}
```



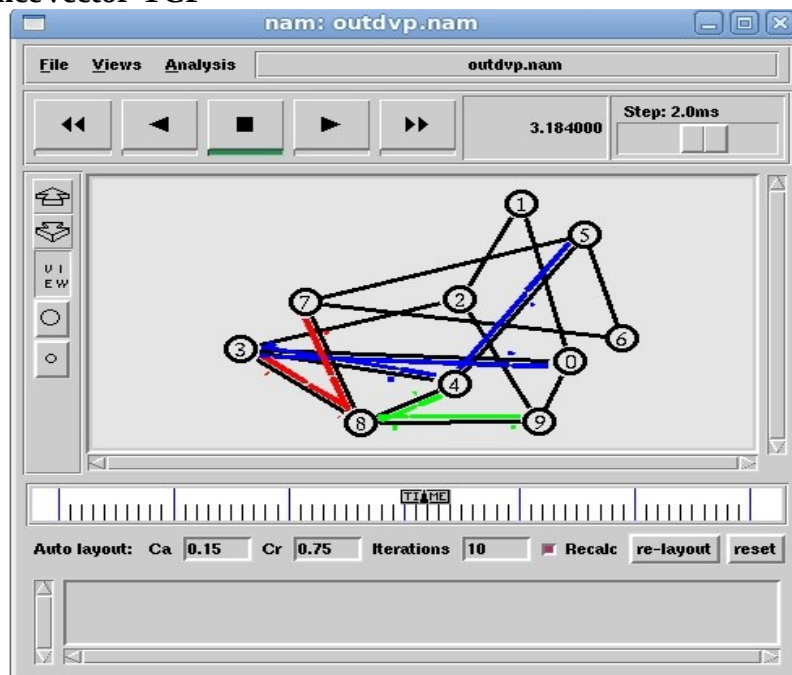
```

# Store received packet's size
recvdSize += pkt_size
}

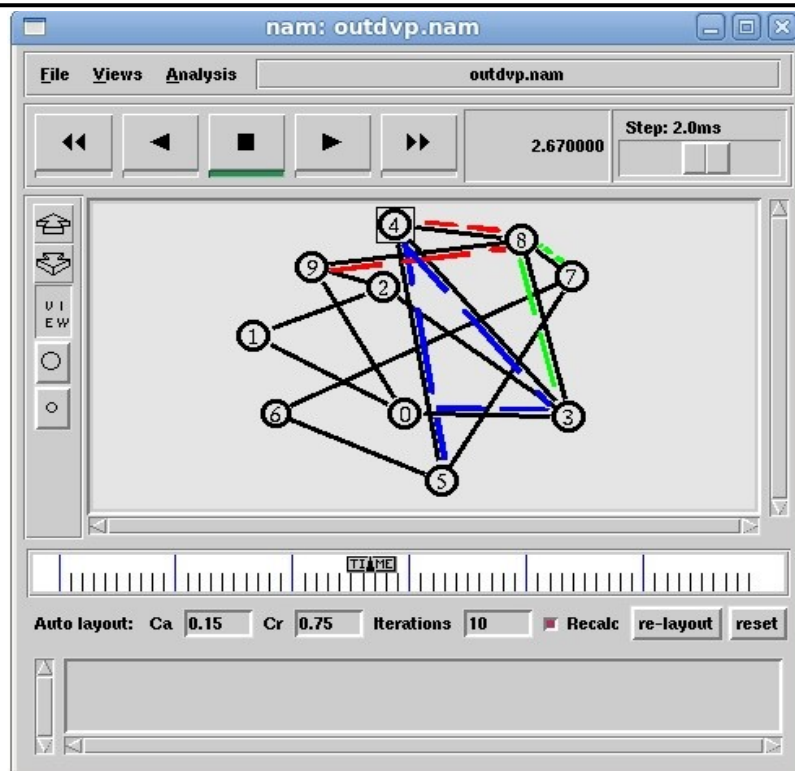
END {
    printf("Average Throughput[kbps] = %.2f\t\t StartTime=
%.2f\tStopTime=%.2f\n", (recvdSize/(stopTime-
startTime))*(8/1000), startTime, stopTime)
}

```

**OUTPUT:**  
**DistanceVector-TCP**



**DistanceVector-UDP**



### RESULT:

Thus the implementation of Distance vector protocol using TCP and UDP connection in NS2 is executed and the throughput and the graph are generated successfully.

EX NO: 8b	LINK STATE ROUTING
DATE:	

**AIM:**

To write a ns2 program for implementing unicast routing 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

//TCP//

```
set ns [new Simulator]
```

```
set nf [open outdvp.nam w]
$ns namtrace-all $nf
```

```
set tr [open outdvp.tr w]
$ns trace-all $tr
```

```
proc finish {} {
    global nf ns tr
    $ns flush-trace
    close $tr
    exec nam outdvp.nam &
    exit 0
}
```

#Define different colors for data flows

```
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green
```

#Create seven nodes

```
for {set i 0} {$i < 10} {incr i} {
    set n($i) [$ns node]
}
```

#Create links between the nodes

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

```
$ns duplex-link $n(0) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(8) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(7) $n(5) 1Mb 10ms DropTail
$ns duplex-link $n(2) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(4) $n(8) 1Mb 10ms DropTail
```

```
$ns rtproto LS
```

```
set tcp0 [new Agent/TCP]
$ns attach-agent $n(0) $tcp0
```

```
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
```

```
#$tcp set class_ 1
```

```
set sink0 [new Agent/TCPSink]
$ns attach-agent $n(5) $sink0
```

```
$ns connect $tcp0 $sink0
```

```
set tcp1 [new Agent/TCP]
$ns attach-agent $n(3) $tcp1
```

```
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
```

```
#$tcp set class_ 1
```

```
set sink1 [new Agent/TCPSink]
$ns attach-agent $n(7) $sink1
```

```
$ns connect $tcp1 $sink1
```

```
set tcp2 [new Agent/TCP]
$ns attach-agent $n(4) $tcp2
```

```
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
```

```
$tcp2 set class_ 1
```

```
set sink2 [new Agent/TCPSink]
$ns attach-agent $n(9) $sink2
```

```
$ns connect $tcp2 $sink2
```

```
$tcp0 set fid_ 1
$tcp1 set fid_ 2
$tcp2 set fid_ 3
```

```
$ns at 0.5 "$ftp0 start"
```

```
$ns at 1.5 "$ftp1 start"  
$ns at 2.5 "$ftp2 start"  
$ns at 3.5 "$ftp0 stop"  
$ns at 4.5 "$ftp1 stop"  
$ns at 5.9 "$ftp2 stop"  
$ns at 6.0 "finish"
```

```
$ns run
```

#### **//TCP AWK FILE**

```
BEGIN {  
    recvdSize = 0  
    startTime = 0  
    stopTime = 0  
}  
  
{  
  
    # Trace line format: new  
    # if ($2 == "-t") {  
        event = $1  
        time = $2  
        pkt_id = $12  
        pkt_size = $6  
        level = $5  
    #}  
  
    # Store start time  
    if (level == "tcp" && (event == "+" || event == "s") && pkt_size >= 500)  
    {  
        if (time < startTime) {  
            startTime = time  
        }  
    }  
  
    # Update total received packets' size and store packets arrival time  
    if (level == "tcp" && event == "r" && pkt_size >= 500) {  
        if (time > stopTime) {  
            stopTime = time  
        }  
        # Rip off the header  
        hdr_size = pkt_size % 500  
        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", (recvSize/(stopTime-
startTime))*(8/1000), startTime, stopTime)
}

//UDP//
set ns [new Simulator]

set nf [open outlsp.nam w]
$ns namtrace-all $nf

set tr [open outlsp.tr w]
$ns trace-all $tr

proc finish {} {
    global nf ns tr
    $ns flush-trace
    close $tr
    exec nam outlsp.nam &
    exit 0
}

#Define different colors for data flows
$ns color 1 Blue
$ns color 2 Red
$ns color 3 Green

#Create seven nodes
for {set i 0} {$i < 10} {incr i} {
    set n($i) [$ns node]
}

#Create links between the nodes
for {set i 0} {$i < 10} {incr i} {
    $ns duplex-link $n($i) $n([expr ($i+1)%10]) 1Mb 10ms DropTail
}

$ns duplex-link $n(0) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(8) $n(3) 1Mb 10ms DropTail
$ns duplex-link $n(7) $n(5) 1Mb 10ms DropTail
$ns duplex-link $n(2) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(4) $n(8) 1Mb 10ms DropTail

$ns rtproto LS

```

```
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
$udp0 set class_ 0
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(4) $udp1
$udp1 set class_ 1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null1 [new Agent/Null]
$ns attach-agent $n(5) $null1
$ns connect $udp1 $null1
```

```
set udp2 [new Agent/UDP]
$ns attach-agent $n(3) $udp2
$udp2 set class_ 2
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 500
$cbr2 set interval_ 0.005
$cbr2 attach-agent $udp2
set null2 [new Agent/Null]
$ns attach-agent $n(7) $null2
$ns connect $udp2 $null2
```

```
$udp0 set fid_ 1
$udp1 set fid_ 2
$udp2 set fid_ 3
```

```
#Schedule events for the CBR agent and the network dynamics
$ns at 0.5 "$cbr0 start"
$ns at 1.5 "$cbr1 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 2.5 "$cbr2 start"
$ns at 3.5 "$cbr0 stop"
$ns at 4.5 "$cbr1 stop"
```



\$ns at 5.9 "\$cbr2 stop"

\$ns at 6.0 "finish"

\$ns run

**//UDP AWK FILE//**

```
BEGIN {
    recvdSize = 0
    startTime = 0
    stopTime = 0
}

{

    # Trace line format: new
    # if ($2 == "-t") {
        event = $1
        time = $2
        pkt_id = $12
        pkt_size = $6
        level = $5
    #}

    # Store start time
    if (level == "cbr" && (event == "+" || event == "s") && pkt_size >= 500)
    {
        if (time < startTime) {
            startTime = time
        }
    }

    # Update total received packets' size and store packets arrival time
    if (level == "cbr" && event == "r" && pkt_size >= 500) {
        if (time > stopTime) {
            stopTime = time
        }
        # Rip off the header
        hdr_size = pkt_size % 500
        pkt_size -= hdr_size
        # Store received packet's size
        recvdSize += pkt_size
    }
}

END {
    printf("Average Throughput[kbps] = %.2f\t\t StartTime=
%.2f\t\t StopTime=%.2f\n", (recvdSize/(stopTime-
startTime))*(8/1000), startTime, stopTime)
}
```

# //GRAPH

Titletext: Throughput Analysis

XUnitText: Number of Nodes

YUnitText: Throughput(Kbps)

Background: white

LineWidth: 5

XLowlimit: 5

XHighlimit: 55

YLowlimit: 2800

YHighlimit: 5100

## "DVP - UDP"

10	2946.69
20	4568.83
30	3443.92
40	4307.74
50	3980.66

## "DVP - TCP"

10	3429.14
20	5022.48
30	3773.38
40	5039.19
50	3256.92

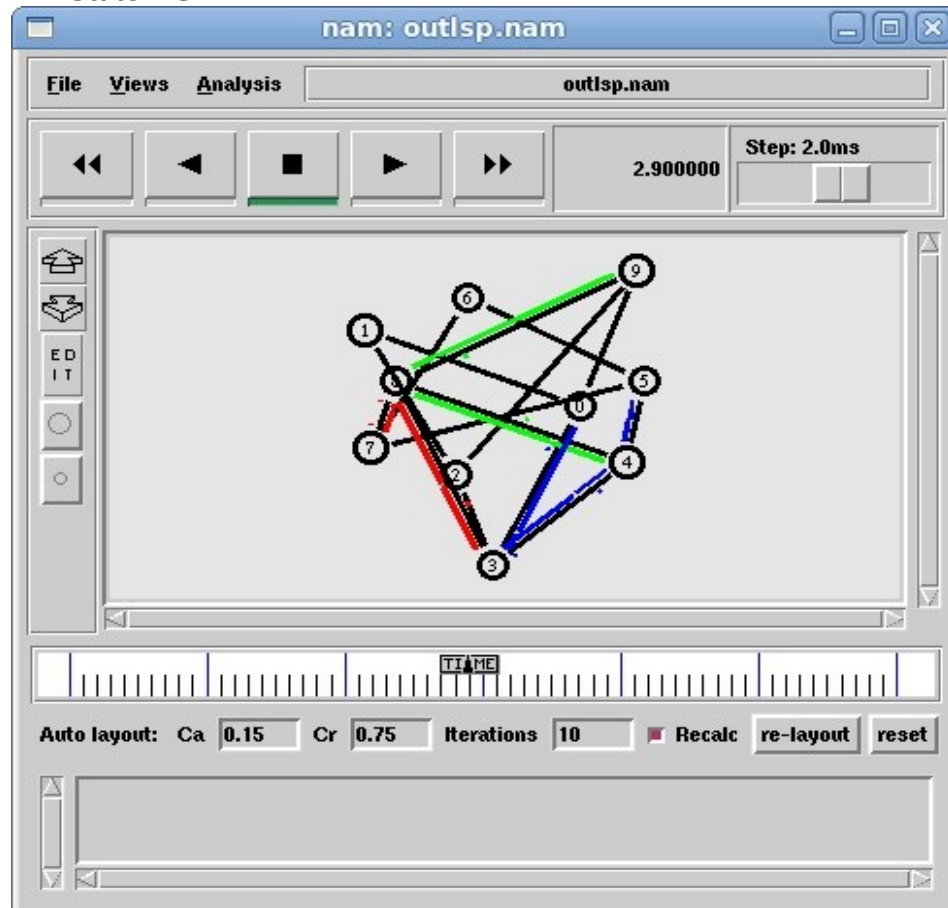
## "LSP - UDP"

10	2946.69
20	3950.57
30	3364.92
40	3849.17
50	3086.77

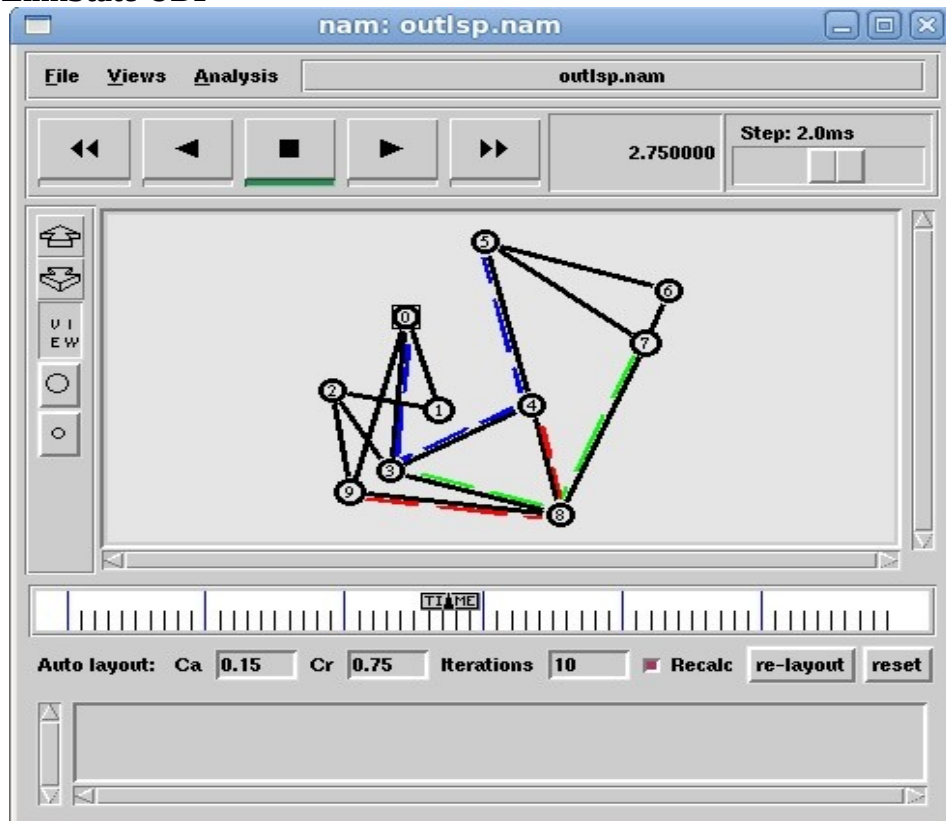
## "LSP - TCP"

10	3429.33
20	4664.95
30	4997.40
40	4504.41
50	3257.14

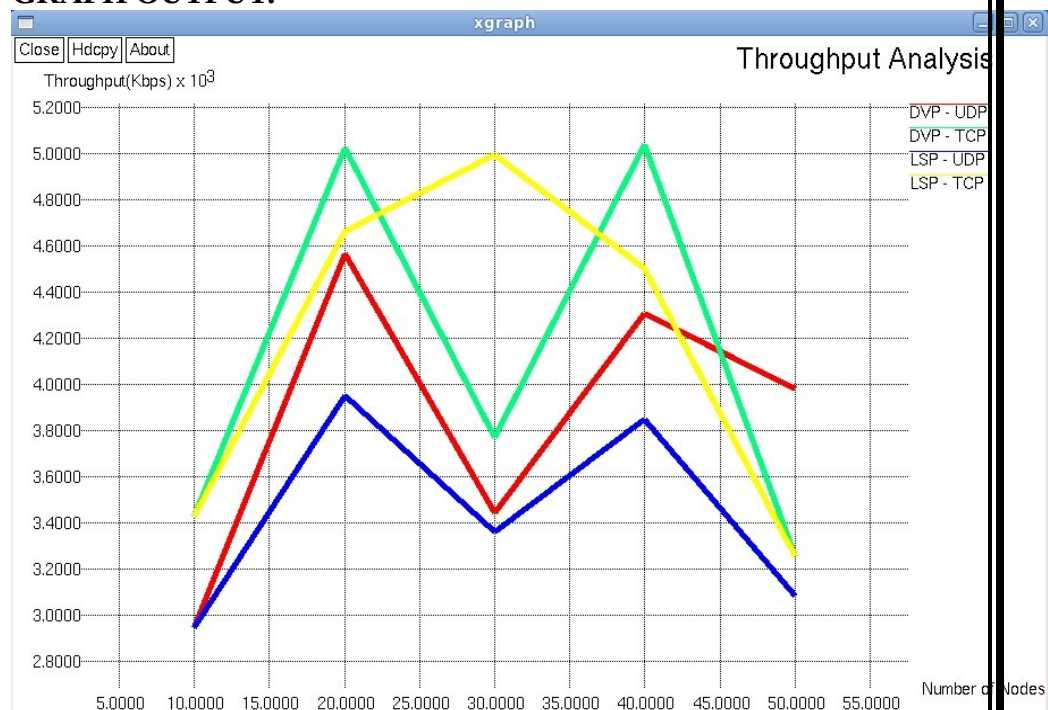
**OUTPUT:**  
**LinkState-TCP**



## LinkState-UDP



## GRAPH OUTPUT:



**RESULT:**

Thus the implementation of Link State protocol using TCP and UDP connection in NS2 is executed and the throughput and the graph are generated successfully.

<b>EX NO: 9</b>	<b>ERROR CORRECTION CODE(LIKE CRC)</b>
<b>DATE:</b>	

**AIM:**

To write a .Java program for Error Correction Code(Like CRC)

**ALGORITHM:**

1. Start the program.
2. Import necessary packages.
3. Get an user input in the form of bit data ie string.
4. Get an generator data from the user.
5. Read the length of the string and convert the data into another format by deducing 48 from it.
6. Now Add the generator data to the original data and send the string as transmitter string.
7. In the receiving end, enter the generator code.
8. Using the generator code, to the length of the received string, add 48 to the number format of the string by character.
9. If the generator string is wrong, display "message received with error".
10. If the generator string is correct, perform step 8 and display "message received with no error".
11. End 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

Data bits are : 1011001

divisor bits are : 101

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.... :)

Press any key to continue...

**Result:**

Thus the Java program for Error Correction Code(Like CRC) is executed successfully.