EX.NO:1(a) Implementation of Sliding Window Protocol

AIM:

To write a java program to perform sliding window protocol

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.net.*; import java.io.*; import
java.rmi.*;
public class slidsender
{
public static void main(String a[])throws Exception
{
ServerSocket ser=new ServerSocket(10); Socket s=ser.accept();
DataInputStream in=new DataInputStream(System.in); DataInputStream in1=new
DataInputStream(s.getInputStream()); String sbuff[]=new String[8];
PrintStream p;
int sptr=0,sws=8,nf,ano,i; String ch;
do
{
p=new PrintStream(s.getOutputStream()); System.out.print("Enter the
no. of frames : "); nf=Integer.parseInt(in.readLine()); p.println(nf);
if(nf<=sws-1)
{

System.out.println("Enter "+nf+" Messages to be send\n"); for(i=1;i<=nf;i++)
{
sbuff[sptr]=in.readLine();
p.println(sbuff[sptr]);
sptr=++sptr%8;
}
sws-=nf;
System.out.print("Acknowledgment received"); ano=Integer.parseInt(in1.readLine());
```

```
System.out.println(" for "+ano+" frames"); sws+=nf;
}
else
{
System.out.println("The no. of frames exceeds window size"); break;
}
System.out.print("\nDo you wants to send some more frames : "); ch=in.readLine(); p.println(ch);
}
while(ch.equals("yes"));
s.close();
}
}
```

RECEIVER PROGRAM

```
import java.net.*; import java.io.*; class slidreceiver
{
public static void main(String a[])throws Exception
{
Socket s=new Socket(InetAddress.getLocalHost(),10); DataInputStream in=new
DataInputStream(s.getInputStream()); PrintStream p=new PrintStream(s.getOutputStream());
int i=0,rptr=-1,nf,rws=8; String rbuf[]=new String[8]; String ch;
System.out.println(); do
{
nf=Integer.parseInt(in.readLine()); if(nf<=rws-1)
{
```

```
for(i=1;i<=nf;i++)
{
rptr=++rptr%8;
rbuf[rptr]=in.readLine();
System.out.println("The received Frame " +rptr+" is : "+rbuf[rptr]);
}
rws-=nf;
System.out.println("\nAcknowledgment
sent\n"); p.println(rptr+1); rws+=nf; }
else
break;
ch=in.readLine();
}
while(ch.equals("yes"));
}
}
```

OUTPUT:

//SENDER OUTPUT

Enter the no. of frames : 4
Enter 4 Messages to be send

hiii
how r u
i am fine
how is evryone
Acknowledgment received for 4 frames

Do you wants to send some more frames : no

//RECEIVER OUTPUT

The received Frame 0 is : hiii
The received Frame 1 is : how r u
The received Frame 2 is : i am fine
The received Frame 3 is : how is everyone

EX.NO:1(b) Implementation of Stop and Wait Protocol

AIM

To write a java program to perform sliding window protocol

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

```
/SENDER PROGRAM

import java.io.*;
import java.net.*;
public class Sender{
Socket sender;
ObjectOutputStream out;
ObjectInputStream in;
String packet,ack,str, msg;
int n,i=0,sequence=0;
Sender(){}
public void run(){
 try{
 BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Waiting for Connection....");
 sender = new Socket("localhost",2004);
sequence=0;

out=new ObjectOutputStream(sender.getOutputStream());
out.flush();
 in=new ObjectInputStream(sender.getInputStream());
 str=(String)in.readObject();
 System.out.println("reciver     > "+str);
System.out.println("Enter the data to send....");
packet=br.readLine();
n=packet.length();
```

```
do{
try{
 if(i<n){
msg=String.valueOf(sequence);
msg=msg.concat(packet.substring(i,i+1));
}
else if(i==n){
 msg="end";out.writeObject(msg);break;
}
out.writeObject(msg);
sequence=(sequence==0)?1:0;
out.flush();
System.out.println("data sent>"+msg);
ack=(String)in.readObject();
System.out.println("waiting for ack.....\n\n");
if(ack.equals(String.valueOf(sequence))){
i++;
System.out.println("receiver   >  "+" packet recieved\n\n");
}
else{
System.out.println("Time out resending data....\n\n");
sequence=(sequence==0)?1:0;
}
}catch(Exception e){}
}while(i<n+1);
System.out.println("All data sent. exiting.");
}catch(Exception e){}
finally{
try{
in.close();
out.close();
sender.close();
}
catch(Exception e){}
}
}
public static void main(String args[]){
Sender s=new Sender();
s.run();
}
}
```

//RECEIVER PROGRAM

```
import java.io.*;
import java.net.*;
public class Reciever{
ServerSocket reciever;
Socket connection=null;
ObjectOutputStream out;
ObjectInputStream in;
String packet,ack,data="";
int i=0,sequence=0;
Reciever(){}
public void run(){
try{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
reciever = new ServerSocket(2004,10);
System.out.println("waiting for connection...");
connection=reciever.accept();
sequence=0;
System.out.println("Connection established   :");
out=new ObjectOutputStream(connection.getOutputStream());
out.flush();
in=new ObjectInputStream(connection.getInputStream());
out.writeObject("connected    .");
do{
try{
packet=(String)in.readObject();
if(Integer.valueOf(packet.substring(0,1))==sequence){
data+=packet.substring(1);
sequence=(sequence==0)?1:0;
System.out.println("\n\nreceiver        >"+packet);
}
else
{
System.out.println("\n\nreceiver        >"+packet +"   duplicate data");
}
if(i<3){
out.writeObject(String.valueOf(sequence));i++;
}
else{
out.writeObject(String.valueOf((sequence+1)%2));
i=0;
}
}
catch(Exception e){}
}while(!packet.equals("end"));
```

```
System.out.println("Data recived="+data);
out.writeObject("connection ended    .");
}
catch(Exception e){}
finally{
try{
in.close();
out.close();
reciever.close();
}
catch(Exception e){}
}
}
public static void main(String args[]){
Reciever s=new Reciever();
while(true){
s.run();
}
}
}
```

OUTPUT:

//SENDER OUTPUT

```
Waiting for Connection....
reciver > connected     .
Enter the data to send....
myname
data sent>0m
waiting for ack.....
receiver       >   packet recieved
data sent>1y
waiting for ack.....
receiver       >   packet recieved
data sent>0n
waiting for ack.....
receiver       >   packet recieved
data sent>1a
waiting for ack.....
Time out resending data....
data sent>1a
waiting for ack.....
receiver       >   packet recieved
data sent>0m
waiting for ack.....
receiver       >   packet recieved
data sent>1e
```

waiting for ack.....
receiver > packet recieved
All data sent. exiting.

//RECEIVER OUTPUT
waiting for connection...
Connection established :
receiver >0m
receiver >1y
receiver >0n
receiver >1a
receiver >1a duplicate data
receiver >0m
receiver >1e
Data recived=myname
waiting for connection...

EX.NO:2 Study of Socket Programming and Client – Server mode

AIM:

To implement socket programming date and time display from client to server using TCP Sockets

ALGORITHM:

Server

1. Create a server socket and bind it to port.
2. Listen for new connection and when a connection arrives, accept it.
3. Send server‟s date and time to the client.
4. Read client‟s IP address sent by the client.
5. Display the client details.
6. Repeat steps 2-5 until the server is terminated.
7. Close all streams.
8. Close the server socket.
9. Stop.

Client

1. Create a client socket and connect it to the server‟s port number.
2. Retrieve its own IP address using built-in function.
3. Send its address to the server.
4. Display the date & time sent by the server.
5. Close the input and output streams.
6. Close the client socket.
7. Stop.

PROGRAM:

DateClient.java:

```
import java.net.*;
import java.io.*;
class dateclient
{
        public static void main (String args[])
        {
                Socket soc;
                DataInputStream dis;
                String sdate;
                PrintStream ps;
                try
                {
                        InetAddress ia=InetAddress.getLocalHost();
                        soc=new Socket(ia,8020);
```

```
                dis=new DataInputStream(soc.getInputStream());
                sdate=dis.readLine();
                System.out.println("THE date in the server is:"+sdate);
                ps=new PrintStream(soc.getOutputStream());
                ps.println(ia);
            }
            catch(IOException e)
            {
                System.out.println("THE EXCEPTION is :"+e);

            }
        }
}
```

DateServer.java:

```
import java.net.*;
import java.io.*;
import java.util.*;
class dateserver
{
    public static void main(String args[])
    {
        ServerSocket ss;
        Socket s;
        PrintStream ps;
        DataInputStream dis;
        String inet;
        try
        {
        ss=new ServerSocket(8020);
        while(true)
        {
        s=ss.accept();
        ps=new PrintStream(s.getOutputStream());
        Date d=new Date();
        ps.println(d);
        dis=new DataInputStream(s.getInputStream());
        inet=dis.readLine();
        System.out.println("THE CLIENT SYSTEM ADDRESS IS :"+inet);
        ps.close();
        }
        }
        catch(IOException e)
        {
            System.out.println("The exception is :"+e);
```

```
                }
        }
}
```

OUTPUT:
CLIENTSIDE:

```
C:\Program Files\Java\jdk1.5.0\bin>javac dateclient.java
Note: dateclient.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
C:\Program Files\Java\jdk1.5.0\bin>java dateclient
THE date in the server is:Sat Mar 19 13:01:16 GMT+05:30 2008
C:\Program Files\Java\jdk1.5.0\bin>
```

SERVERSIDE:

```
C:\Program Files\Java\jdk1.5.0\bin>javac dateserver.java
Note: dateserver.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
C:\Program Files\Java\jdk1.5.0\bin>java dateserver
THE CLIENT SYSTEM ADDRESS IS :com17/192.168.21.17
```

EX.NO:3 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 (3(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

EX-NO. 4(a). Write a code simulating PING command

Aim:

To Write the java program for simulating ping command

Algorithm

Step 1: start the program.
Step 2: Include necessary package in java.
Step 3: To create a process object p to implement the ping command.
Step 4: declare one BufferedReader stream class object.
Step 5: Get thedetails of the server
    5.1: length of the IP address.
    5.2: time required to get the details.
    5.3: send packets , receive packets and lost packets.
    5.4: minimum ,maximum and average times.
Step 6: print the results.
Step 7:Stop the program.

Program:

```
import java.io.*;
import java.net.*;
class pingserver
{
public static void main(String args[])
{
try
{
String str;
System.out.print(" Enter the IP Address to be Ping : ");
BufferedReader buf1=new BufferedReader(new
InputStreamReader(System.in));
String ip=buf1.readLine();
Runtime H=Runtime.getRuntime();
Process p=H.exec("ping " + ip);
InputStream in=p.getInputStream();
BufferedReader buf2=new BufferedReader(new
InputStreamReader(in));
while((str=buf2.readLine())!=null)
{
System.out.println(" " + str);
}
}
catch(Exception e)
{
```

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

Output:

```
Enter the IP address to the ping:192.168.0.1

Pinging 192.168.0.1: with bytes of data =32

Reply from 192.168.0.11:bytes=32 time<1ms TTL =128
Reply from 192.168.0.11:bytes=32 time<1ms TTL =128
Reply from 192.168.0.11:bytes=32 time<1ms TTL =128
Reply from 192.168.0.11:bytes=32 time<1ms TTL =128

Ping statistics for 192.168.0.1
Packets: sent=4,received=4,lost=0(0% loss),approximate round trip time in milli seconds:
Minimum=1
ms,maximum=4ms,average=2ms
```

EX-NO. 4(b). Write a code simulating TRACEROUTE command

Aim:

To Write the java program for simulating Traceroute command

Algorithm

Program

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import traceroute.TracerouteItem;

public abstract class Traceroute
{

        private Runtime run;

    public Traceroute()
    {
        run = Runtime.getRuntime();
    }

    public ArrayList<TracerouteItem> traceroute(String destination)
    {

  ArrayList<TracerouteItem> result = new ArrayList<TracerouteItem>();

                Process pr = null;

        String cmd = getTracerouteCommand(destination);

        try
        {
            pr = run.exec(cmd);
        }
        catch(IOException e)
        {
                        e.printStackTrace();
        }

      BufferedReader buf = new BufferedReader(new InputStreamReader(
                pr.getInputStream()));

                String line = "";
```

```
            try
            {
                  while((line = buf.readLine()) != null)
                  {
                        TracerouteItem item = parse(line);
                        result.add(item);
                  }
            }
            catch(IOException e)
            {                 return null;
            }

                  return result;
      }

            public abstract TracerouteItem parse(String line);

      public abstract String getTracerouteCommand(String destination);

}
```

Output

EX.NO 5. 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.

EX-NO 6. Write a program to implement RPC(Remote Procedure Call)

Aim:
To write a java program to implement RPC (remote procedure call)

Algorithm :

Step 1: start the program.
Step 2: include the necessary packages in java.
Step 3: create an interface as ucet and extends the predefined interface remote into it.
Step 4: declare the remote methods inside the interface used.
Step 5: declare the class rpc and extends the predefined class.
Step 6: unicast remote object an implement the interface ucet into it.
Step 7: declare the class server &create an object ob for the class rpc.
Step 8: declare the class client
Step 9: call the remote methods using the object ob which is the object of the interface ucet.
Step 10: the remote method contains the methods such as functions(a,b),power(a,b)and log(a).
Step 11:Stop the program.

Program:

Client:

```
import java.rmi.*;
import java.io.*;
import java.rmi.server.*;
public class clientrpc
{
public static void main(String arg[])
{
try
{
String serverurl="rmi://localhost/serverrpc";
ucet ob=(ucet) Naming.lookup(serverurl);
int r=ob.function(10,5);
System.out.println("the answer of(a+b)^2 is:"+r);
int t =ob.power(10,5);
System.out.println("the answer of(a)^(b) is:"+t);
double d=ob.log(10);
System.out.println("the log value of the given number "+10+" is :"+d);
}
catch(Exception e)
{
System.out.println("error.."+e.getMessage());
}
}
}
```

Server:

```
import java.rmi.*;
import java.rmi.server.*;
public class serverrpc
{
public static void main(String arg[])
{
try
{
rpc ob=new rpc();
Naming.rebind("serverrpc",ob);
}
catch(Exception e)
{
}
}}
```

RPC:

```
import java.rmi.*;
import java.lang.Math.*;
import java.rmi.server.*;
public class rpc extends UnicastRemoteObject implements ucet
{
public rpc()throws Exception
{
}
public int function(int a,int b)throws RemoteException
{
int m;
m=(a*a)+(b*b)+(2*a*b);
return m;
}
public int power(int a,int b)throws RemoteException
{
int m=(int)Math.pow(a,b);
return m;
}
public double log(int a)throws RemoteException
{
return(Math.log(a));
}
}
```

Ucet:

```
import java.rmi.*;
public interface ucet extends Remote
{
public int function(int a,int b)throws RemoteException;
public int power(int a,int b)throws RemoteException;
```

```
public double log(int a)throws RemoteException;
}
```

Output:

```
Javac  ucet.java
Start  rmiregistry

Javac  rpc.java
rmi  rpc
javac clientrpc.java
javac serverrpc.java
javac rpc.java
javac ucet.java

javac serverrpc.java
java serverrpc

javac clientrpc.java
java clientrpc
the ans of (a+b)^2 is:225
the ans of (a)^(b) is :100000
the log value of the given number 10 is 2.30258
```

EX-NO 7. Implementation of Subnetting

Aim:

Write a program to implement subnetting and find the subnet masks.

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.util.Scanner;
class Subnet
{
public static void main(String args[])

{
Scanner sc = new Scanner(System.in);
System.out.print("Enter the ip address");
String ip = sc.nextLine();
String split_ip[]=ip.split("\\.");

 //SPlit the string after every .
String split_bip[] = new String[4];

//split binary ip
String bip = "";
for(int i=0;i<4;i++){
split_bip[i] = appendZeros(Integer.toBinaryString(Integer.parseInt(split_ip[i])));

// "18" => 18 => 10010=>00010010
bip += split_bip[i];
}
System.out.println("IP in binary is "+bip);
System.out.print("Enter the number of addresses: ");
int n = sc.nextInt();
```

```
//Calculation of mask

int bits = (int)Math.ceil(Math.log(n)/Math.log(2));

/*eg if address = 120, log 120/log 2 gives log to the base 2 => 6.9068, ceil gives us upper
integer */

System.out.println("Number of bits required for address = "+bits); int mask = 32-bits;
System.out.println("The subnet mask is = " +mask);

//Calculation of first address and last

address int fbip[] = new int[32];
for(int i=0; i<32;i++) fbip[i] = (int)bip.charAt(i)-48;

//convert cahracter 0,1 to integer 0,1 for(int i=31;i>31-bits;i–)
//Get first address by ANDing last n bits with 0

fbip[i] &= 0;
String fip[] = {"","","","",""};
for(int i=0;i<32;i++)
fip[i/8] = new String(fip[i/8]+fbip[i]);
System.out.print("First address is = ");
for(int i=0;i<4;i++){
System.out.print(Integer.parseInt(fip[i],2));
if(i!=3) System.out.print(".");
}
System.out.println();
int lbip[] = new int[32];
for(int i=0; i<32;i++) lbip[i] = (int)bip.charAt(i)-48; //convert cahracter 0,1 to integer 0,1
for(int i=31;i>31-bits;i–)       //Get last address by ORing last n bits with 1
lbip[i] |= 1;
String lip[] ={"","","","",""};
for(int i=0;i<32;i++)
lip[i/8] = new String(lip[i/8]+lbip[i]);
System.out.print("Last address is =");
 for(int i=0;i<4;i++){
System.out.print(Integer.parseInt(lip[i],2));
if(i!=3) System.out.print(".");
}
System.out.println();
}
static String appendZeros(String s){
String temp = new String("00000000″);
return temp.substring(s.length())+ s;
}
}
```

Output

```
Enter the ip address: 100.110.150.10
IP in binary is 01100100011011101001011000001010
Enter the number of addresses: 7
Number of bits required for address = 3
The subnet mask is = 29
First address is = 100.110.150.8
Last address is = 100.110.150.15
```

EX-NO 8. 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();
            c=new Socket(ia,9000);
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
        try
        {
            os=new PrintStream(c.getOutputStream());
            is=new DataInputStream(System.in);
            is1=new DataInputStream(c.getInputStream());
            while(true)
            {
            System.out.println("Client:");
                line=is.readLine();
                os.println(line);
                System.out.println("Server:" + is1.readLine());
            }}

        catch(IOException e)
        {
            System.out.println("Socket Closed!");
```

```
                }
        }
}
```

Output

Server

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

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

jbsdava
jbvuesagv

EX-NO 9. Applications using TCP and UDP Sockets like DNS, SNMP and File Transfer

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();
}
}
```

OUTPUT

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
```

B.SNMP

Aim

To write a java program for SNMP application program

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.IOException;
import org.snmp4j.CommunityTarget;
import org.snmp4j.PDU;
 import org.snmp4j.Snmp;
 import org.snmp4j.Target;
 import org.snmp4j.TransportMapping;
 import org.snmp4j.event.ResponseEvent;
 import org.snmp4j.mp.SnmpConstants;
 import org.snmp4j.smi.Address;
 import org.snmp4j.smi.GenericAddress;
 import org.snmp4j.smi.OID;
 import org.snmp4j.smi.OctetString;
 import org.snmp4j.smi.VariableBinding;
 import org.snmp4j.transport.DefaultUdpTransportMapping;

 public class SNMPManager {


    Snmp snmp = null;
    String address = null;

    * Constructor

    * @param
    add
*/
```

```
public SNMPManager(String add)
{
address = add;
public static void main(String[] args) throws IOException {

/**
* Port 161 is used for Read and Other operations
* Port 162 is used for the trap
   generation */
   SNMPManager client = new SNMPManager("udp:127.0.0.1/161");
client.start();
/**
* OID - .1.3.6.1.2.1.1.1.0 => SysDec
* OID - .1.3.6.1.2.1.1.5.0 => SysName
   * => MIB explorer will be usefull here, as discussed in previous
   article */

String sysDescr = client.getAsString(new OID(".1.3.6.1.2.1.1.1.0"));
System.out.println(sysDescr);
}

   /**
* get any answers because the communication is asynchronous
* and the listen() method listens for answers.
* @throws IOException
*/
   private void start() throws IOException {
   TransportMapping transport = new DefaultUdpTransportMapping();
snmp = new
Snmp(transport);
// Do not forget this line!
transport.listen();
}

/**
* Method which takes a single OID and returns the response from the agent as a String.
* @param oid
* @return
* @throws IOException
*/
```

```
public String getAsString(OID oid) throws IOException {

ResponseEvent event = get(new OID[] { oid });
return event.getResponse().get(0).getVariable().toString();
}
/**
* This method is capable of handling multiple OIDs
* @param oids
* @return
* @throws IOException
*/
public ResponseEvent get(OID oids[]) throws IOException {
PDU pdu = new PDU();
for (OID oid : oids) {
pdu.add(new VariableBinding(oid));
}
pdu.setType(PDU.GET);
ResponseEvent event = snmp.send(pdu, getTarget(), null);
 if(event != null) {
return event;
}
throw new RuntimeException("GET timed out");
}
/**
* This method returns a Target, which contains information about
* where the data should be fetched and how.
* @return
*/
private Target getTarget() {
Address targetAddress = GenericAddress.parse(address);
CommunityTarget target = new CommunityTarget();
target.setCommunity(new OctetString("public"));
 target.setAddress(targetAddress);
target.setRetries(2);
target.setTimeout(1500);
target.setVersion(SnmpConstants.version2c);
 return target;
}}
```

OUT PUT

Hardware: x86 Family 6 Model 23 Stepping 10 AT/AT COMPATIBLE – Software: Windows 2000 Version 5.1 (Build 2600 Multiprocessor Free)

C. File Transfer

AIM

To write a java program for applaction using TCP and UDP Sockets Links

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

jbsdava jbvuesagv

EX-NO 10. 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 (ns-2.26, Feb 2003)
- Platform support
- FreeBSD, Linux, Solaris, Windows and Mac

Ns unctionalities

Routing, Transportation, Traffic sources,Queuing disciplines, QoS

Wireless

Ad hoc routing, mobile IP, sensor-MAC
Tracing, visualization and various utilitie
NS(Network Simulators)

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

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

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

Examples of network simulators

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

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

Uses of network simulators

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

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

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

Packet loss

occurs when one or morepacketsof data travelling across a computer networkfail to reachtheir destination. Packet loss is distinguished as one of the three main error types encountered in digital communications; the other two being bit errorand spurious packets caused due to noise.

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

Throughput

This is the main performance measure characteristic, and most widely used. Incommunicationnetworks, such asEthernetorpacket radio, throughputor network throughputis the average rate of successfulmessage delivery over a communication channel. The throughput is usually measured inbitsper second (bit/s orbps), andsometimes indata packetsper second or data packets pertime slotThis measure how soon the receiver is able to get a certain amount of data send by the sender. It is determined as the ratio of the total data received to the end to end delay. Throughput is an important factor which directly impacts the network performance

Delay

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

Queue Length

A queuing system in networks can be described as packets arriving for service, waiting for service if it is not immediate, and if having waited for service, leaving thesystem 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.

11. Perform a case study about the different routing algorithms to select the

network path with its optimum and economical during data transfer.

i. **Link State routing**

Aim:

To study the link state routing

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

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

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 neighbours 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

- f 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's protocols) to calculate paths.

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

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

Examples of distance-vector routing protocols include RIPv1 and RIPv2 and IGRP.

Method

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

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

Distance-vector protocols are based on calculating the direction and distance to any link in a network. "Direction" usually means the next hop address and the exit interface. "Distance" is a

measure of the cost to reach a certain node. The least cost route between any two nodes is the route with minimum distance. Each node maintains a vector (table) of minimum distance to every node. The cost of reaching a destination is calculated using various route metrics. RIP uses the hop count of the destination whereas IGRP takes into account other information such as node delay and available bandwidth.

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

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

Count-to-infinity problem

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

Ex.no.11 UNICAST ROUTING PROTOCOL

Aim:

To write a ns2 program for implementing unicast routing protocol.

Algorithm:

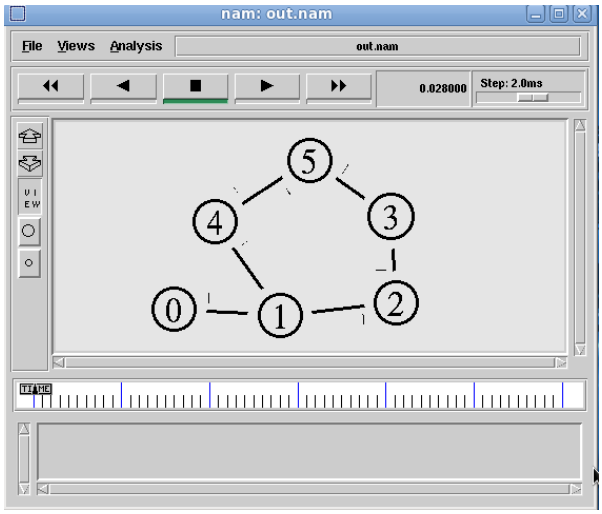
Step 1: start the program.
Step 2: declare the global variables ns for creating a new simulator.
Step 3: set the color for packets.
Step 4: open the network animator file in the name of file2 in the write mode.
Step 5: open the trace file in the name of file 1 in the write mode.
Step 6: set the unicast 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.

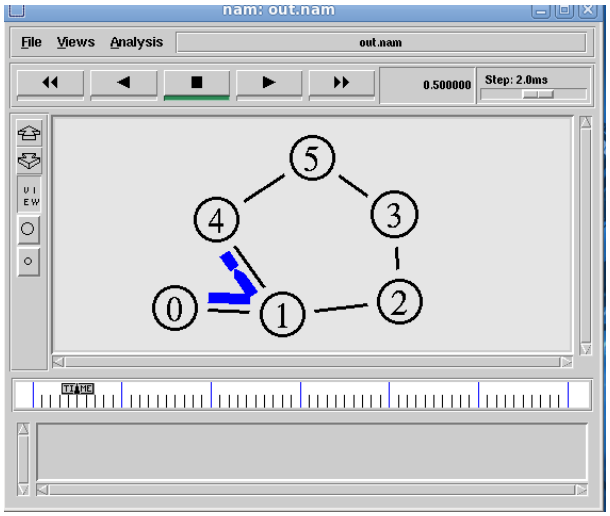
Program:

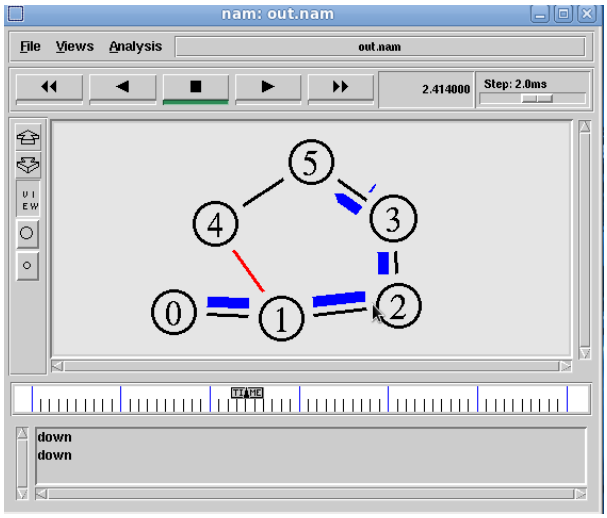
```
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the Trace file
set file1 [open out.tr w]
$ns trace-all $file1
```

```
#Open the NAM trace file
set file2 [open out.nam w]
$ns namtrace-all $file2

#Define a 'finish' procedure
proc finish {} {
        global ns file1 file2
        $ns flush-trace
        close $file1
        close $file2
        exec nam out.nam &
        exit 3
}

# Next line should be commented out to have the static routing
$ns rtproto DV

#Create six nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n4 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n1 0.3Mb 10ms DropTail
$ns duplex-link $n1 $n2 0.3Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.3Mb 10ms DropTail
$ns duplex-link $n1 $n4 0.3Mb 10ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 10ms DropTail
$ns duplex-link $n4 $n5 0.5Mb 10ms DropTail

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


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

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

$ns rtmodel-at 1.0 down $n1 $n4
$ns rtmodel-at 4.5 up $n1 $n4

$ns at 0.1 "$ftp start"

$ns at 6.0 "finish"

$ns run
```







Ex.no.11(b) MULTICASTING ROUTING PROTOCOL

Aim:

To write a ns2 program for implementing multicasting 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 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.

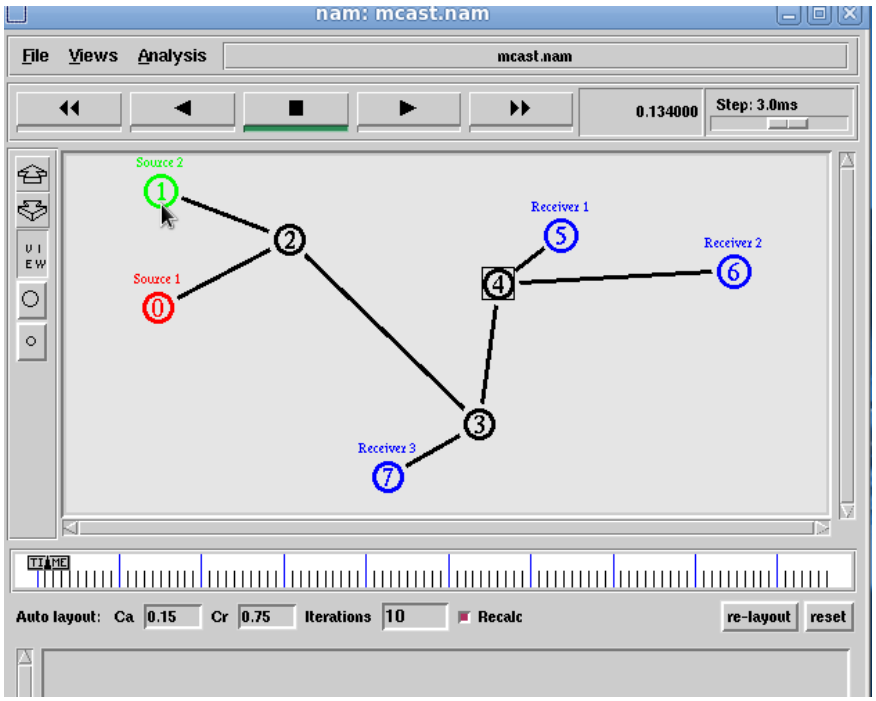
Program:

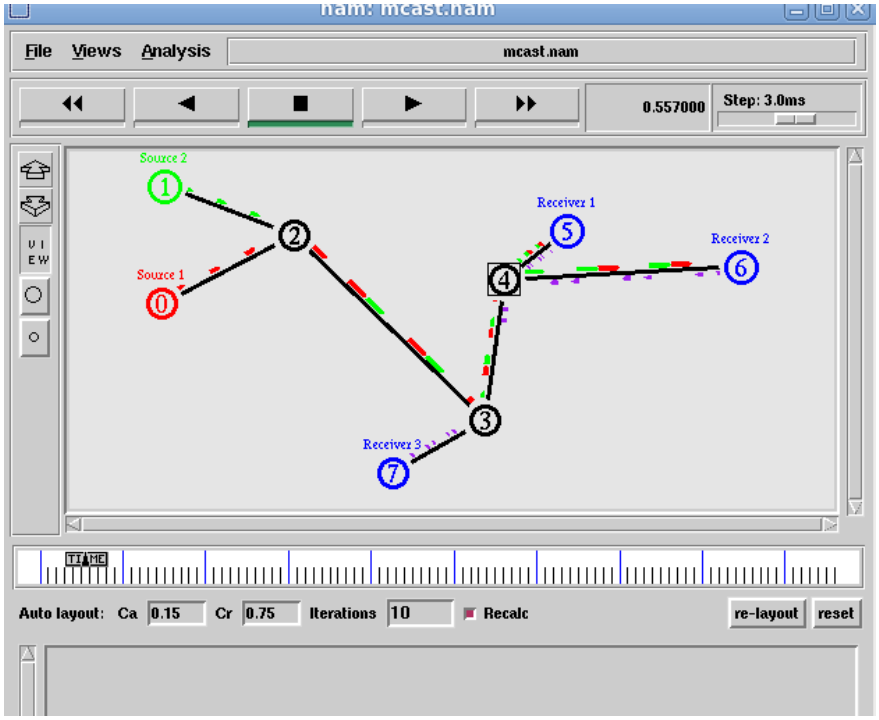
```
# Create scheduler
#Create an event scheduler wit multicast turned on
set ns [new Simulator -multicast on]

#$ns multicast
#Turn on Tracing
set tf [open output.tr w]
$ns trace-all $tf

# Turn on nam Tracing
set fd [open mcast.nam w]
$ns namtrace-all $fd

# Create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]

# Create links
$ns duplex-link $n0 $n2 1.5Mb 10ms DropTail
```

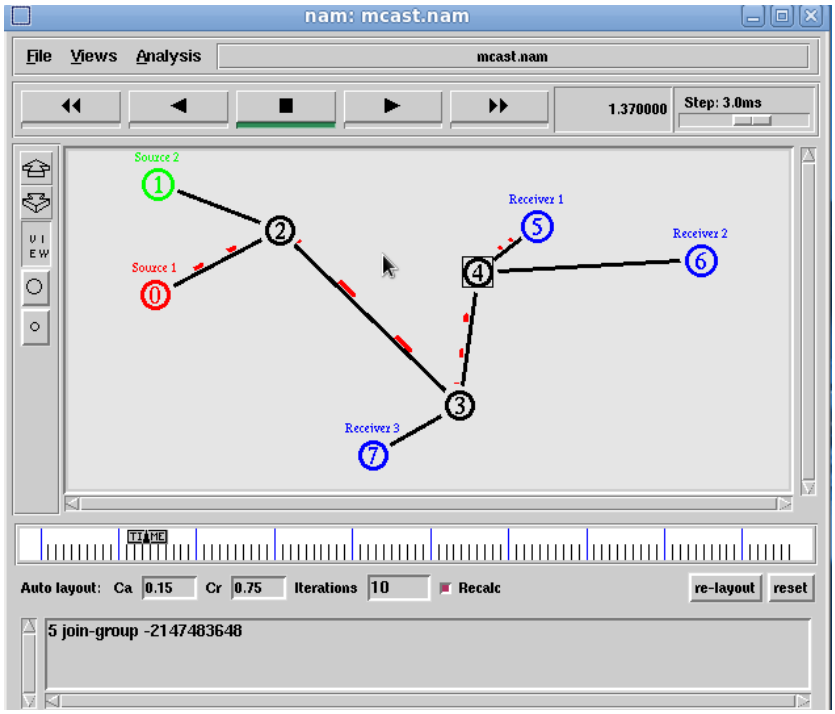
```
$ns duplex-link $n1 $n2 1.5Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
$ns duplex-link $n3 $n4 1.5Mb 10ms DropTail
$ns duplex-link $n3 $n7 1.5Mb 10ms DropTail
$ns duplex-link $n4 $n5 1.5Mb 10ms DropTail
$ns duplex-link $n4 $n6 1.5Mb 10ms DropTail

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

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

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

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

# Create receiver
set rcvr1 [new Agent/Null]
$ns attach-agent $n5 $rcvr1
$ns at 1.0 "$n5 join-group $rcvr1 $group1"
set rcvr2 [new Agent/Null]
$ns attach-agent $n6 $rcvr2
$ns at 1.5 "$n6 join-group $rcvr2 $group1"
set rcvr3 [new Agent/Null]
$ns attach-agent $n7 $rcvr3
$ns at 2.0 "$n7 join-group $rcvr3 $group1"
set rcvr4 [new Agent/Null]
$ns attach-agent $n5 $rcvr1
$ns at 2.5 "$n5 join-group $rcvr4 $group2"
set rcvr5 [new Agent/Null]
$ns attach-agent $n6 $rcvr2
$ns at 3.0 "$n6 join-group $rcvr5 $group2"
set rcvr6 [new Agent/Null]
$ns attach-agent $n7 $rcvr3
```

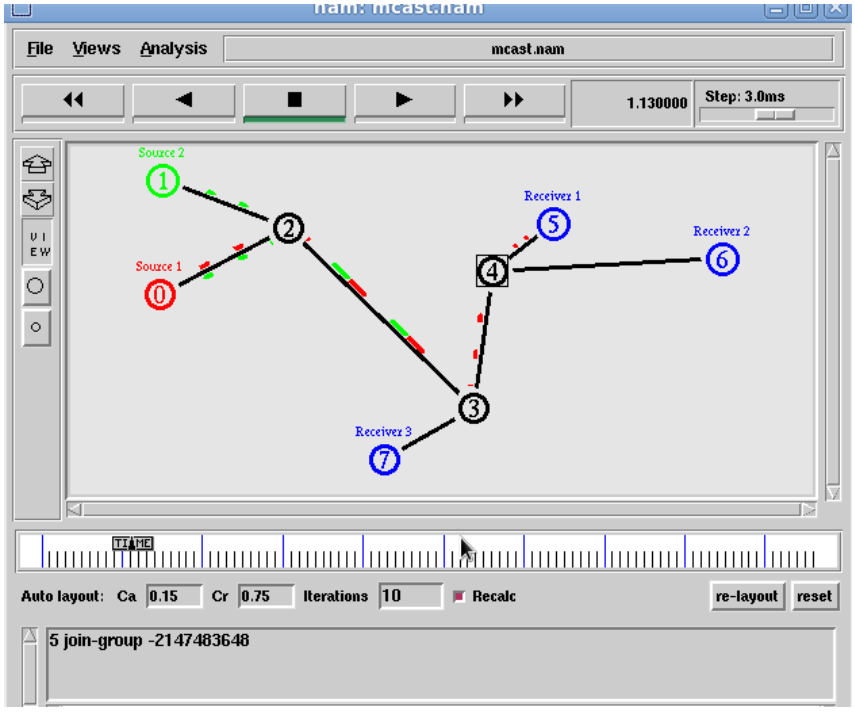
```
$ns at 3.5 "$n7 join-group $rcvr6 $group2"
$ns at 4.0 "$n5 leave-group $rcvr1 $group1"
$ns at 4.5 "$n6 leave-group $rcvr2 $group1"
$ns at 5.0 "$n7 leave-group $rcvr3 $group1"
$ns at 5.5 "$n5 leave-group $rcvr4 $group2"
$ns at 6.0 "$n6 leave-group $rcvr5 $group2"
$ns at 6.5 "$n7 leave-group $rcvr6 $group2"

# Schedule events
$ns at 0.5 "$cbr1 start"
$ns at 9.5 "$cbr1 stop"
$ns at 0.5 "$cbr2 start"
$ns at 9.5 "$cbr2 stop"

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

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


# For nam
#Colors for packets from two mcast groups
$ns color 10 red
$ns color 11 green
$ns color 30 purple
$ns color 31 green

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

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

# Group 1 source
$udp1 set fid_ 11
$n1 color green
$n1 label "Source 2"
$n5 label "Receiver 1"
```

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

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

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









Ex.no.11(c) CARRIER SENSE MULTIPLE ACCESS

Aim:

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

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 write mode.
Step 5: open the trace file and the win file in the write mode.
Step 6: transfer the packets in network.
Step 7: create the 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 tcp connection for source node.
Step 11: set the destination node using tcp sink.
Step 12: set the window size and the packet size for the tcp.
Step 13: set up the ftp over the tcp connection.
Step 14:.set the udp and tcp connection for the source and destination.
Step 15: create the traffic generator CBR for the source and destination files.
Step 15: define the plot window and 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]
$ns color 1 blue
$ns color 2 red
set fi1 [open out.tr w]
set winfile [open WinFile w]
$ns trace-all $fi1
set fi2 [open out.nam w]
$ns namtrace-all $fi2
proc finish {} {
global ns fi1 fi2
$ns flush-trace
close $fi1
close $fi2
exec nam out.nam &
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n1 color red
$n1 shape box
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set window_ 8000
$tcp set packetsize_ 552
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2
set cbr [new Application/Traffic/CBR]
```

```
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 0.01mb
$cbr set random_ false
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 24.0 "$ftp stop"
$ns at 24.5 "$cbr stop"
proc plotwindow { tcpSource file } {
global ns
set time 0.1
set now [$ns now]
set cwnd [$tcpSource set cwnd_]
set wnd [$tcpSource set window_]
puts $file "$now $cwnd"
$ns at [expr $now+$time] "plotwindow $tcpSource $file"
}
$ns at 1.0 "plotwindow $tcp $winfile"
$ns at 5 "$ns trace-annotate \"packet drop\""
$ns at 125.0 "finish"
$ns run
```

Output: