EXNO: 1	NETWORKING COMMANDS
DATE:	

EXP NO 1

AIM

To study the basic networking commands.

COMMANDS

C:\>arp –a: ARP is short form of address resolution protocol, It will show the IP address of your computer along with the IP address and MAC address of your router.

C:\>hostname: This is the simplest of all TCP/IP commands. It simply displays the name of your computer.

C:\>ipconfig: The ipconfig command displays information about the host (the computer your sitting at)computer TCP/IP configuration.

C:\>ipconfig /all: This command displays detailed configuration information about your TCP/IP connection including Router, Gateway, DNS, DHCP, and type of Ethernet adapter in your system.

C:\>Ipconfig /renew: Using this command will renew all your IP addresses that you are currently (leasing) borrowing from the DHCP server. This command is a quick problem solver if you are having connection issues, but does not work if you have been configured with a static IP address.

C:\>Ipconifg /release: This command allows you to drop the IP lease from the DHCP server.

C:\>ipconfig /flushdns: This command is only needed if you're having trouble with your networks DNS configuration. The best time to use this command is after network configuration frustration sets in, and you really need the computer to reply with flushed.

C:\>nbtstat –a: This command helps solve problems with NetBIOS name resolution. (Nbt stands for NetBIOS over TCP/IP)

C:\>netdiag: Netdiag is a network testing utility that performs a variety of network diagnostic tests, allowing you to pinpoint problems in your network. Netdiag isn't installed by default, but can be installed from the Windows XP CD after saying no to the install. Navigate to the CD ROM drive letter and open the support\tools folder on the XP CD and click the setup.exe icon in the support\tools folder.

C:\>netstat: Netstat displays a variety of statistics about a computers active TCP/IP connections. This tool is most useful when you're having trouble with TCP/IP applications such as HTTP, and FTP.

C:\>nslookup: Nslookup is used for diagnosing DNS problems. If you can access a resource by specifying an IP address but not it's DNS you have a DNS problem.

C:\>pathping: Pathping is unique to Window's, and is basically a combination of the Ping and Tracert commands. Pathping traces the route to the destination address then launches a 25 second test of each router along the way, gathering statistics on the rate of data loss along each hop.
C:\>ping: Ping is the most basic TCP/IP command, and it's the same as placing a phone call to your best friend. You pick up your telephone and dial a number, expecting your best friend to reply with "Hello" on the other end. Computers make phone calls to each other over a network by using a Ping command. The Ping commands main purpose is to place a phone call to another computer on the network, and request an answer. Ping has 2 options it can use to place a phone call to another computer on the network. It can use the computers name or IP address.

C:\>route: The route command displays the computers routing table. A typical computer, with a single network interface, connected to a LAN, with a router is fairly simple and generally doesn't pose any network problems. But if you're having trouble accessing other computers on your network, you can use the route command to make sure the entries in the routing table are correct.

C:\>tracert: The tracert command displays a list of all the routers that a packet has to go through to get from the computer where tracert is run to any other computer on the internet.

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

Out put :

```
Welcome to network programming world
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
```

RESULT

Thus the above list of primitive has been studied.

EXNO: 2	WEB PAGE DOWNLOADING
DATE:	

AIM

To download a webpage using Java

ALGORITHM:

CLIENT SIDE:

1) Start the program.
2) Create a socket which binds the Ip address of server and the port address to acquire service.
3) After establishing connection send the url to server.
4) Open a file and store the received data into the file.
5) Close the socket.
6) End the program.

SERVER SIDE

1) Start the program.
2) Create a server socket to activate the port address.
3) Create a socket for the server socket which accepts the connection.
4) After establishing connection receive url from client.
5) Download the content of the url received and send the data to client.
6) Close the socket.
7) End the program.

PROGRAM

```
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 PROGRAM

```
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

RESULT

The webpage is successfully downloaded and the contents are displayed and verified.

EXNO: 3A	SOCKET PROGRAM FOR ECHO
DATE:	

AIM

To write a socket program for implementation of echo.

ALGORITHM

CLIENT SIDE

1. Start the program.
2. Create a socket which binds the Ip address of server and the port address to acquire service.
3. After establishing connection send a data to server.
4. Receive and print the same data from server.
5. Close the socket.
6. End the program.

SERVER SIDE

1. Start the program.
2. Create a server socket to activate the port address.
3. Create a socket for the server socket which accepts the connection.
4. After establishing connection receive the data from client.
5. Print and send the same data to client.
6. Close the socket.
7. End the program.

PROGRAM

ECHO CLIENT

```
import java.io.*;
import java.net.*;
public class eclient
{
public static void main(String args[])
{
Socket c=null;
String line;
DataInputStream is,is1;
PrintStream os;
```

```
try
{
c=new Socket("localhost",8080);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
os=new PrintStream(c.getOutputStream());
is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
do
{
System.out.println("client");
line=is.readLine();
os.println(line);
if(!line.equals("exit"))
System.out.println("server:"+is1.readLine());
}while(!line.equals("exit"));
}
catch(IOException e)
{
System.out.println("socket closed");
}}}
```

Echo Server:

```
import java.io.*;
import java.net.*;
import java.lang.*;
public class eserver
{
public static void main(String args[])throws IOException
{
ServerSocket s=null;
String line;
```

```
DataInputStream is;
PrintStream ps;
Socket c=null;
try
{
s=new ServerSocket(8080);
}
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();
System.out.println("msg received and sent back to client");
ps.println(line);
}
}
catch(IOException e)
{
System.out.println(e);
}
}
}
```

OUTPUT

CLIENT

Enter the IP address 127.0.0.1

CONNECTION ESTABLISHED

Enter the data SRM

Client received SRM

SERVER

CONNECTION ACCEPTED

Server received SRM

RESULT

Thus the program for simulation of echo server was written & executed

EXNO: 3B	CLIENT- SERVER APPLICATION FOR CHAT
DATE:	

AIM

To write a client-server application for chat using TCP

ALGORITHM

CLIENT

1. Start the program
2. Include necessary package in java
3. To create a socket in client to server.
4. The client establishes a connection to the server.
5. The client accept the connection and to send the data from client to server.
6. The client communicates the server to send the end of the message
7. Stop the program.

SERVER

1. Start the program
2. Include necessary package in java
3. To create a socket in server to client
4. The server establishes a connection to the client.
5. The server accept the connection and to send the data from server to client and
6. vice versa
7. The server communicate the client to send the end of the message.
8. Stop the program.

PROGRAM

TCPserver1.java

```
import java.net.*;
import java.io.*;
public class TCPserver1
{
 public static void main(String arg[])
{
ServerSocket s=null;
```

```
String line;
DataInputStream is=null,is1=null;
PrintStream os=null;
Socket c=null;
try
{
s=new ServerSocket(9999);
}
catch(IOException e)
{
System.out.println(e);
 }
try
{
c=s.accept();
is=new DataInputStream(c.getInputStream());
is1=new DataInputStream(System.in);
os=new PrintStream(c.getOutputStream());
do
{
line=is.readLine();
System.out.println("Client:"+line);
System.out.println("Server:");
line=is1.readLine();
os.println(line);
}
while(line.equalsIgnoreCase("quit")==false);
is.close();
os.close();
}
catch(IOException e)
{
System.out.println(e);
}}}
```

TCPclient1.java

```
import java.net.*;
import java.io.*;
 public class TCPclient1
{
 public static void main(String arg[])
{
Socket c=null;
String line;
DataInputStream is,is1;
PrintStream os;
try
{
c=new Socket("10.0.200.36",9999);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
 os=new PrintStream(c.getOutputStream());
is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
do
{
System.out.println("Client:");
line=is.readLine();
os.println(line);
System.out.println("Server:" + is1.readLine());
}
while(line.equalsIgnoreCase("quit")==false);
is1.close();
os.close();
```

```
}
catch(IOException e)
{
System.out.println("Socket Closed!Message Passing is over");
}}
```

OUTPUT:

SERVER

```
C:\Program Files\Java\jdk1.5.0\bin>javac TCPserver1.java
Note: TCPserver1.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
C:\Program Files\Java\jdk1.5.0\bin>java TCPserver1

Client: Hai Server
Server:Hai Client
Client: How are you
Server:Fine
Client: quit
Server:quit
```

CLIENT

```
C:\Program Files\Java\jdk1.5.0\bin>javac TCPclient1.java
Note: TCPclient1.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.
C:\Program Files\Java\jdk1.5.0\bin>java TCPclient1

Client:Hai Server
Server: Hai Client
Client:How are you
Server: Fine
Client:quit
Server: quit
```

RESULT

Thus the above program a client-server application for chat using TCP / IP was executed and successfully.

EXNO: 3C	FILE TRANSFER IN CLIENT & SERVER
DATE:	

AIM

To Perform File Transfer in Client & Server Using TCP/IP.

ALGORITHM

CLIENT SIDE

1. Start.
2. Establish a connection between the Client and Server.
3. Socket ss=new Socket(InetAddress.getLocalHost(),1100);
4. Implement a client that can send two requests.
   i) To get a file from the server.
   ii) To put or send a file to the server.
5. After getting approval from the server ,the client either get file from the server or send file to the server.

SERVER SIDE

1. Start.
2. Implement a server socket that listens to a particular port number.
3. Server reads the filename and sends the data stored in the file for the‘get’ request.
4. It reads the data from the input stream and writes it to a file in theserver for the ‘put’ instruction.
5. Exit upon client’s request.
6. Stop.

PROGRAM

CLIENT SIDE

```
import java.net.*;
import java.io.*;
public class FileClient{
  public static void main (String [] args ) throws IOException {
    int filesize=6022386; // filesize temporary hardcoded
    long start = System.currentTimeMillis();
    int bytesRead;
    int current = 0;
    // localhost for testing
```

```
   Socket sock = new Socket("127.0.0.1",13267);
   System.out.println("Connecting...");
   // receive file
   byte [] mybytearray = new byte [filesize];
   InputStream is = sock.getInputStream();
   FileOutputStream fos = new FileOutputStream("source-copy.pdf");
   BufferedOutputStream bos = new BufferedOutputStream(fos);
   bytesRead = is.read(mybytearray,0,mybytearray.length);
   current = bytesRead;
   // thanks to A. Cádiz for the bug fix
   do {
     bytesRead =
        is.read(mybytearray, current, (mybytearray.length-current));
     if(bytesRead >= 0) current += bytesRead;
   } while(bytesRead > -1);

   bos.write(mybytearray, 0 , current);
   bos.flush();
   long end = System.currentTimeMillis();
   System.out.println(end-start);
   bos.close();
   sock.close();
 }}
```

SERVER SIDE

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

public class FileServer
{
  public static void main (String [] args ) throws IOException {
    ServerSocket servsock = new ServerSocket(13267);
    while (true) {
      System.out.println("Waiting...");
      Socket sock = servsock.accept();
```

```
System.out.println("Accepted connection : " + sock);
File myFile = new File ("source.pdf");
byte [] mybytearray = new byte [(int)myFile.length()];
FileInputStream fis = new FileInputStream(myFile);
BufferedInputStream bis = new BufferedInputStream(fis);
bis.read(mybytearray,0,mybytearray.length);
OutputStream os = sock.getOutputStream();
System.out.println("Sending...");
os.write(mybytearray,0,mybytearray.length);
os.flush();
sock.close();
}}}
```

OUTPUT

SERVER OUTPUT

C:\Program Files\Java\jdk1.6.0\bin>javac FServer.java

C:\Program Files\Java\jdk1.6.0\bin>java FServer

Waiting for clients...

Connection Established

Client wants file:network.txt

CLIENT OUTPUT

C:\Program Files\Java\jdk1.6.0\bin>javac FClient.java

C:\Program Files\Java\jdk1.6.0\bin>java FClient

Connection request.....Connected

Enter the filename: network.txt

Computer networks: A computer network, often simply referred to as a network, is a collection of computers and devices connected by communications channels that facilitates communications among users and allows users to share resources with other user

RESULT

Thus the File transfer Operation is done & executed successfully

EXNO: 4	Simulation of DNS using UDP sockets
DATE:	

Aim

To write a java program for DNS 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

//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**

```
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

Result

Thus the simulation of DNS using UDP sockets done & executed successfully.

EXNO: 5	Simulating ARP /RARP protocols
DATE:	

AIM

To implement Address Resolution Protocol .

ALGORITHM

CLIENT SIDE

1. Establish a connection between the Client and Server.
   Socket ss=new Socket(InetAddress.getLocalHost(),1100);
2. Create instance output stream writer
   PrintWriter ps=new PrintWriter(s.getOutputStream(),true);
3. Get the IP Address to resolve its physical address.
4. Send the IPAddress to its output Stream.ps.println(ip);
5. Print the Physical Address received from the server.

SERVER SIDE

1. Accept the connection request by the client.
   ServerSocket ss=new ServerSocket(2000);Socket s=ss.accept();
2. Get the IPaddress from its inputstream.
   BufferedReader br1=new BufferedReader(newInputStreamReader(s.getInputStream()));
   ip=br1.readLine();
3. During runtime execute the processRuntime r=Runtime.getRuntime();
   Process p=r.exec("arp -a "+ip);
4. Send the Physical Address to the client.

PROGRAM

ARP CLIENT

```
import java.io.*;
import java.net.*;
class ArpClient
{
public static void main(String args[])throws IOException
{
  try
{
Socket ss=new Socket(InetAddress.getLocalHost(),1100);
PrintStream ps=new PrintStream(ss.getOutputStream());
```

```
BufferedReader br=new BufferedReader(newInputStreamReader(System.in));
String ip;
System.out.println("Enter the IPADDRESS:");
ip=br.readLine();
ps.println(ip);
String str,data;
BufferedReader br2=new BufferedReader(newInputStreamReader(ss.getInputStream()));
System.out.println("ARP From Server::");
do
{
str=br2.readLine();
System.out.println(str);
}
while(!(str.equalsIgnoreCase("end")));
}
catch(IOException e)
{
System.out.println("Error"+e);
}}}
```

ARP SERVER

```
import java.io.*;
import java.net.*;
class ArpServer
{
public static void main(String args[])throws IOException
{
try
{
ServerSocket ss=new ServerSocket(1100);
Socket s=ss.accept();
PrintStream ps=new PrintStream(s.getOutputStream());
BufferedReader br1=new BufferedReader(newInputStreamReader(s.getInputStream()));
String ip;
 ip=br1.readLine();
```

```
Runtime r=Runtime.getRuntime();
Process p=r.exec("arp -a "+ip);
BufferedReader br2=new BufferedReader(newInputStreamReader(p.getInputStream()));
String str;
while((str=br2.readLine())!=null)
{
ps.println(str);
}}
catch(IOException e)
{
System.out.println("Error"+e);  }}}
```

OUTPUT

```
C:\Networking Programs>java ArpServer
C:\Networking Programs>java ArpClient
Enter the IPADDRESS:
192.168.11.58
ARP From Server:
Interface: 192.168.11.57 on Interface 0x1000003
Internet Address      Physical Address        Type
192.168.11.58         00-14-85-67-11-84       dynamic
```

RESULT

Thus the implementation of ARP is done & executed successfully.

EXNO: 6	Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS
DATE:	

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

NS Functionalities

Routing, Transportation, Traffic sources,

Queuing disciplines, QoS

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 describe the state of the network (nodes, routers, switches, and links) and the events (data transmissions, packet error etc.). An important output of simulations is 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 variants" 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

It 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 send by the sender. It is determined as the ratio of the total data received to the end to end delay. Throughput is an important factor which directly impacts the network performance

Delay

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

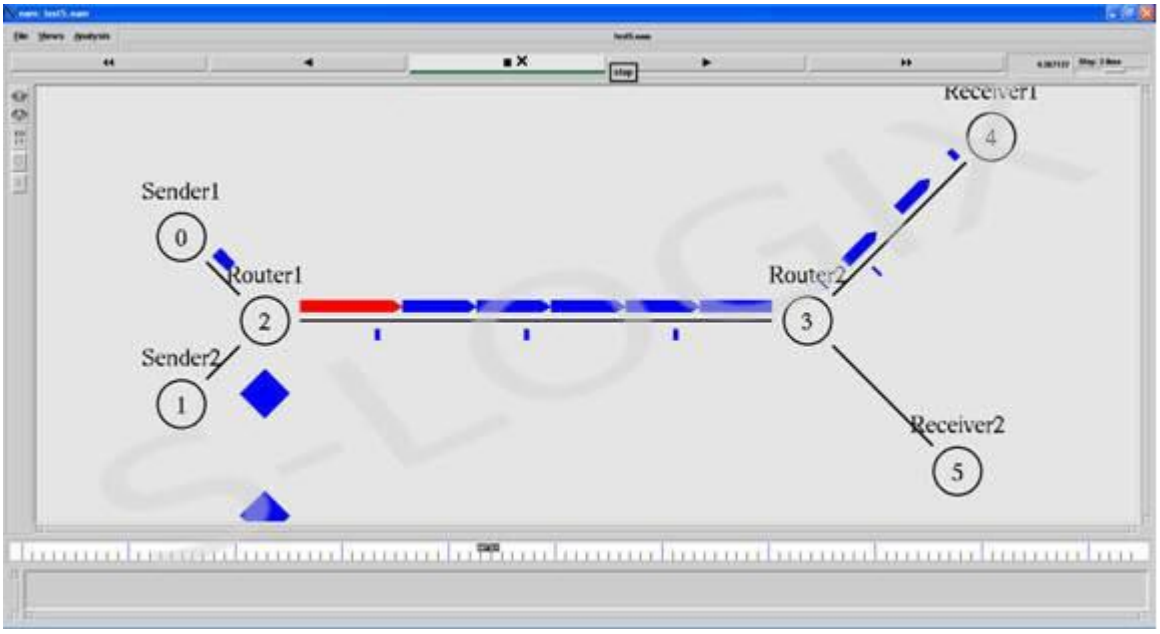
Congestion Control Algorithms

The topology in the wired network is set-up using the node and link creation APIs. The bottleneck is a duplex link that has router in both directions representing the traffic flow from multiple sources. The congestion in the wired network can be created using the bottleneck link. The tcl script in test5.tcl creates the congestion in which each link is configured with the specific bandwidth, propagation delay and DropTail queue. Data transmission between the sender1 and receiver1 is created using the tcp connection and FTP application. Data transmission between the sender2 and receiver2 is created using the udp agent and CBR traffic. Congestion occurs at the link between router1 and router2 (bottleneck link) due to the high data flow rate that exceeds the link capacity. The length of the queue is limited. TCP enforces congestion control mechanism automatically by adjusting the sending data rate according to the acknowledgment arrival time.

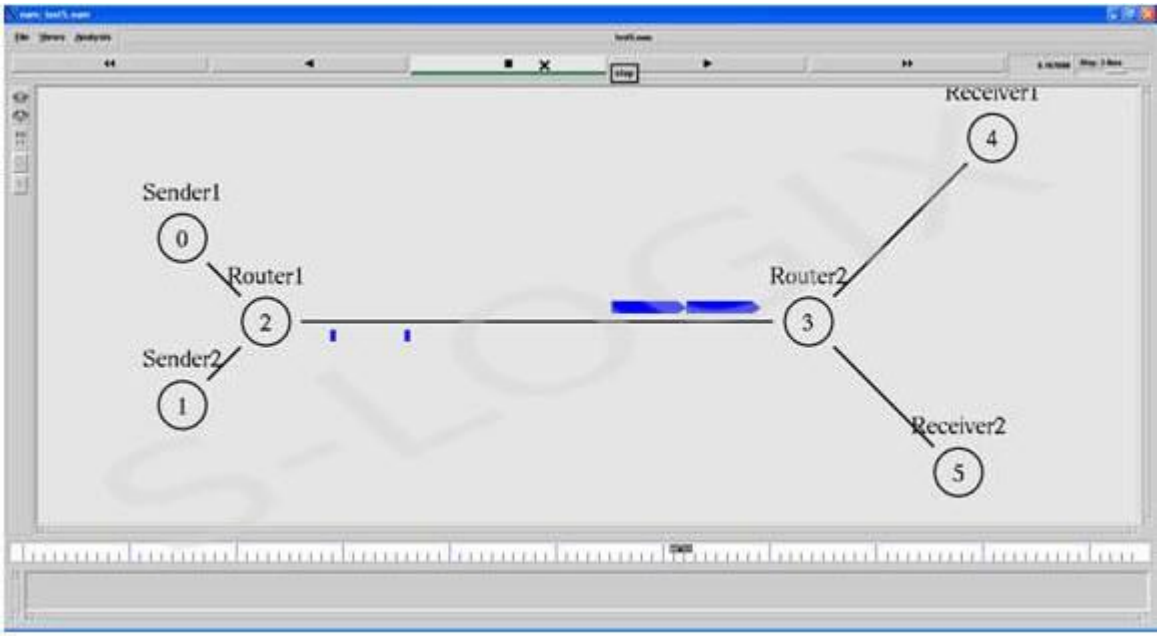
```
# Filename: test5.tcl
#create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
$ns duplex-link $n3 $n4 0.5Mb 40ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 30ms DropTail
#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 20
#Setup a TCP connection
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
#Setup a FTP over TCP connection
set ftp [new Application/FTP]
```

```
$ftp attach-agent $tcp
$ftp set type_ FTP
#Setup a UDP connection
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
#Setup a CBR over UDP connection
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"
$ns at 10.0 "$ftp stop"
$ns at 10.5 "$cbr stop"
```

Congestion



Congestion control mechanism of TCP



RESULT

Thus the Network simulator (NS2) was studied and the performance of congestion control using NS2 code has been implemented.

EXNO: 7	TCP/UDP MODULE IMPLEMENTATION
DATE:	

AIM

To write a socket program for implementation of TCP module.

ALGORITHM

CLIENT SIDE

1) Start the program.
2) Create a socket which binds the Ip address of server and the port address to acquire service.
3) After establishing connection send a data to server.
4) Close the socket.
5) End the program.

SERVER SIDE

1) Start the program.
2) Create a server socket to activate the port address.
3) Create a socket for the server socket which accepts the connection.
4) After establishing connection receive the data from client.
5) Print the data.
6) Close the socket.
7) End the program.

SERVER PROGRAM

```
 import java.io.*;
import java.net.*;
public class server1
{
public static void main(String args[])throws IOException
{
ServerSocket s=new ServerSocket(8080);
System.out.println("socket is created");
System.out.println("waiting for client");
Socket s1=s.accept();
DataOutputStream d1=new DataOutputStream(s1.getOutputStream());
BufferedReader c=new BufferedReader(new InputStreamReader(System.in));
String e=c.readLine();
d1.writeUTF(e);
d1.close();
s1.close();
}}
```

CLIENT PROGRAM

```
import java.io.*;
import java.net.*;
public class client2
{
static String a;
static String b;
public static void main(String args[])throws IOException
{

Socket s=new Socket("127.0.0.1",8080);
BufferedReader c=new BufferedReader(new InputStreamReader(System.in));
DataOutputStream d1=new DataOutputStream(s.getOutputStream());
DataInputStream d2=new DataInputStream(s.getInputStream());
do
{
String st=new String(d2.readUTF());
System.out.println("s::"+st);
System.out.println("c::");
a=c.readLine();
d1.writeUTF(a);
}while(!a.equals("exit"));
d2.close();
d1.close();
    lose();
}
}
```

OUTPUT

CLIENT

Enter the IP address 127.0.0.1
CONNECTION ESTABLISHED
Enter the data SRM

SERVER
CONNECTION ACCEPTED
Server received SRM

RESULT

Thus the program for TCP module implementation was written & executed.

EXNO: 8a	Simulation of Distance Vector algorithm.
DATE:	

AIM:

To simulate and study the Distance Vector routing algorithm using simulation.

THEORY:

Distance Vector Routing is one of the routing algorithm in a Wide Area Network for computing shortest path between source and destination. The Router is one main devices used in a wide area network. The main task of the router is Routing. It forms the routing table and delivers the packets depending upon the routes in the table-either directly or via an intermediate devices.

Each router initially has information about its all neighbors. Then this information will be shared among nodes.

ALGORITHM:

1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose distance vector routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

PROGRAM:

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

$ns namtrace-all $nf
proc finish { } {
global ns nr nf
      $ns flush-trace
close $nf
close $nr
exec nam thro.nam &
        exit 0
```

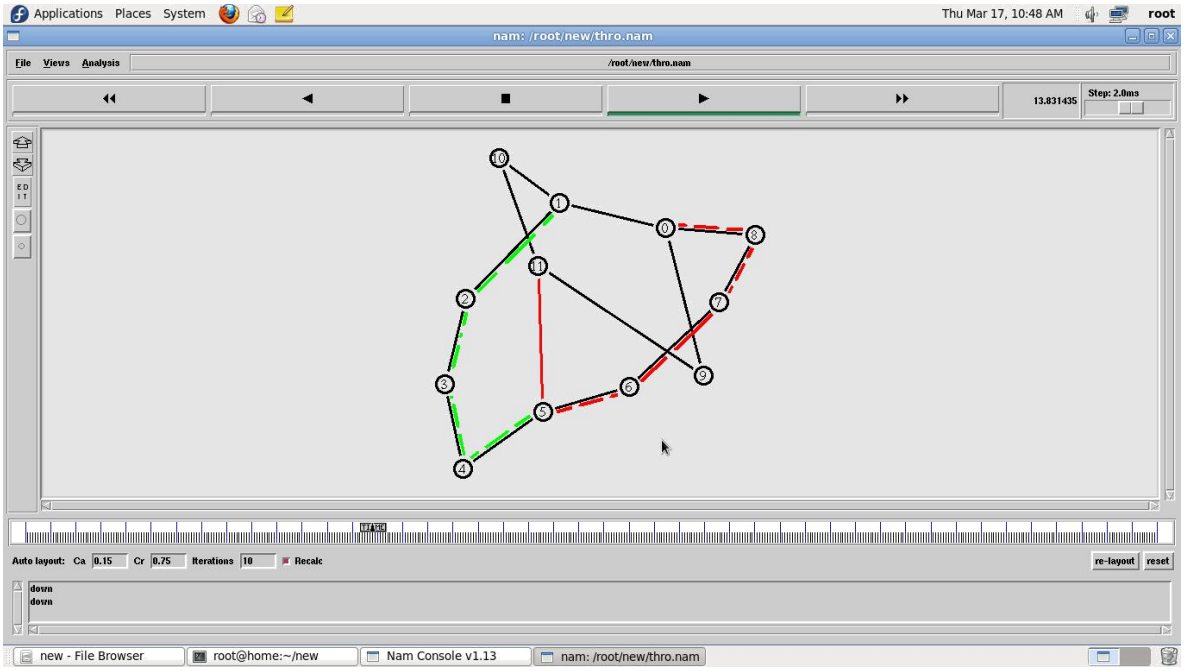
```
    }

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

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

$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb10msDropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail


set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0


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

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

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

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

$ns at 45 "finish"
$ns run

OUTPUT:



RESULT:

Thus the Distance vector Routing Algorithm was Simulated and studied.

EXNO: 8b	Simulation of Link State Routing algorithm.
DATE:	

AIM

To simulate and study the link state routing algorithm using simulation.

THEORY:

In **link state routing,** each router shares its knowledge of its neighborhood with every other router in the internet work. (i) **Knowledge about Neighborhood:** Instead of sending its entire routing table a router sends info about its neighborhood only. (ii) **To all Routers:** each router sends this information to every other router on the internet work not just to its neighbor .It does so by a process called **flooding. (iii)Information sharing when there is a change:** Each router sends out information about the neighbors when there is change.

PROCEDURE:

The Dijkstra algorithm follows four steps to discover what is called the **shortest path tree**(routing table) for each router:The algorithm begins to build the tree by identifying its roots. The root router's trees the router itself. The algorithm then attaches all nodes that can be reached from the root. The algorithm compares the tree's temporary arcs and identifies the arc with the lowest cumulative cost. This arc and the node to which it connects are now a permanent part of the shortest path tree. The algorithm examines the database and identifies every node that can be reached from its chosen node. These nodes and their arcs are added temporarily to the tree.

The last two steps are repeated until every node in the network has become a permanent part of the tree.

ALGORITHM:

1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes

6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose Link state routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

PROGRAM:

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

$ns namtrace-all $nf
proc finish { } {
global ns nr nf
      $ns flush-trace
close $nf
close $nr
exec nam thro.nam &
         exit 0
      }
for { set i 0 } { $i < 12} { incr i 1 } {
set n($i) [$ns node]}

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

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

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

set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0

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

$ns rtproto LS
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)

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

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

$ns color 1 Red

$ns color 2 Green

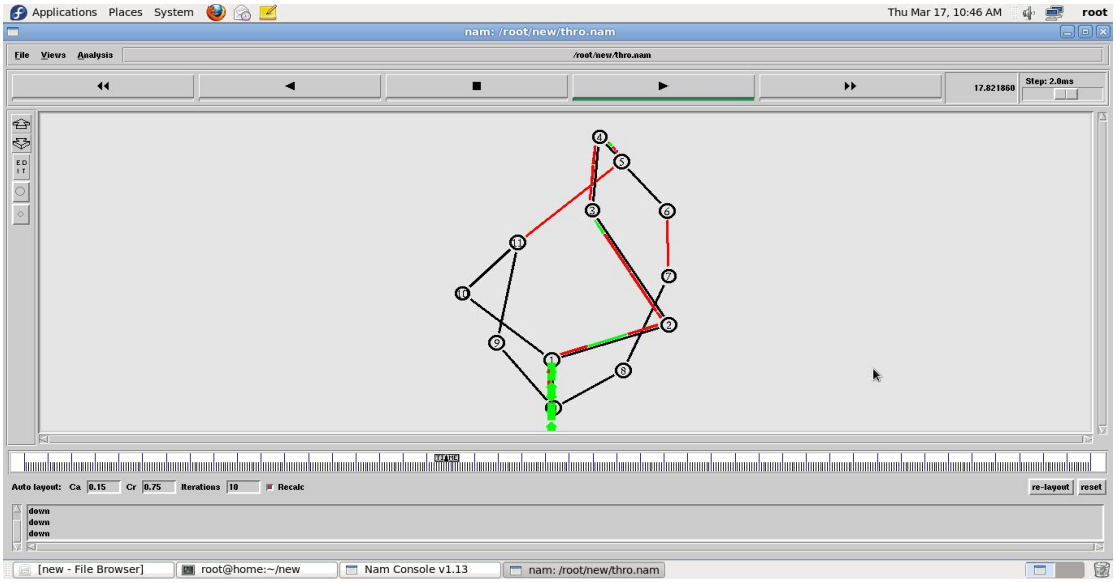
$ns at 1.0 "$cbr0 start"

$ns at 2.0 "$cbr1 start"

$ns at 45 "finish"

$ns run

OUTPUT:



RESULT:

Thus the Link State Routing Algorithm was Simulated and studied

EXNO: 9	**Performance evaluation of Routing protocols using Simulation tool**
DATE:	

AIM:

To Simulate and to study of Go Back N protocol

THEORY:

Go Back N is a connection oriented transmission. The sender transmits the frames continuously. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size. The sender has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously. The size of the window depends on the protocol designer.

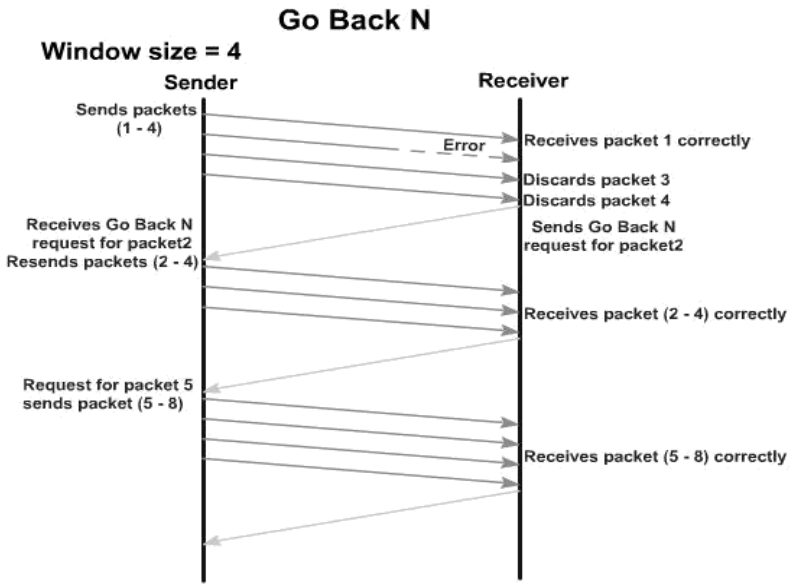
OPERATIONS:

1. A station may send multiple frames as allowed by the window size.
2. Receiver sends an ACK i if frame i has an error. After that, the receiver discards all incoming frames until the frame with error is correctly retransmitted.
3. If sender receives an ACK i it will retransmit frame i and all packets i+1, i+2,... which have been sent, but not been acknowledged

ALGORITHM FOR GO BACK N

1. The source node transmits the frames continuously.
2. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size.
3. The source node has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously.
4. The size of the window depends on the protocol designer.
5. For the first frame, the receiving node forms a positive acknowledgement if the frame is received without error.
6. If subsequent frames are received without error (up to window size) cumulative positive acknowledgement is formed.
7. If the subsequent frame is received with error, the cumulative acknowledgment error-free frames are transmitted. If in the same window two frames or more frames are received with error, the second and the subsequent error frames are neglected. Similarly even the frames received without error after the receipt of a frame with error are neglected.
8. The source node retransmits all frames of window from the first error frame.
9. If the frames are errorless in the next transmission and if the acknowledgment is error free, the window slides by the number of error-free frames being transmitted.

10. If the acknowledgment is transmitted with error, all the frames of window at source are retransmitted, and window doesn't slide.
11. This concept of repeating the transmission from the first error frame in the window is called as **GOBACKN** transmission flow control protocol.



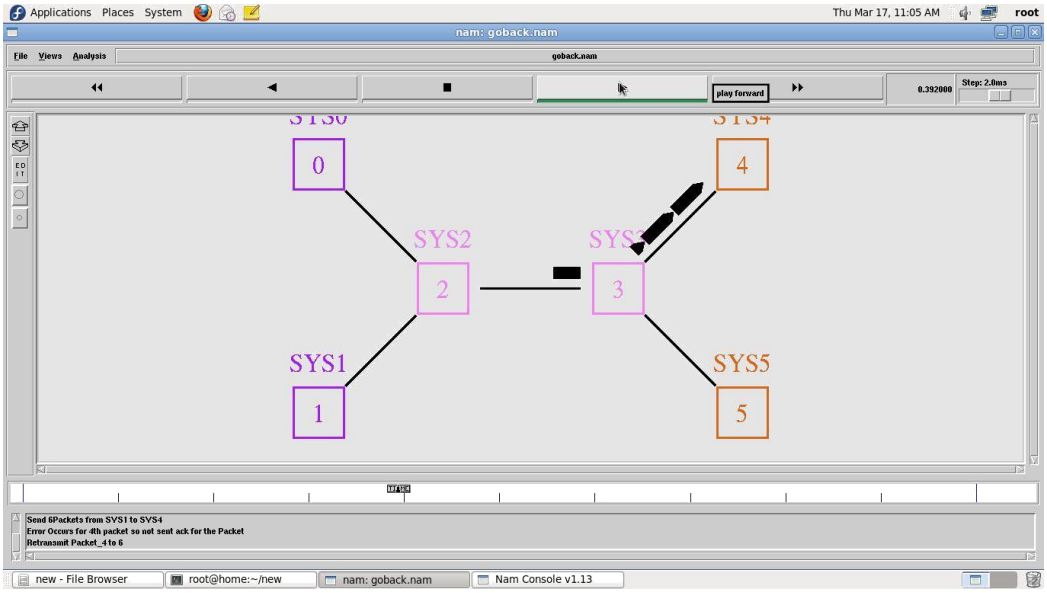
 # PROGRAM FOR GOBACK N:

```
#  #send packets one by one
# send packets one by one
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5[$ns node] $n0 color
"purple" $n1 color
"purple" $n2 color
"violet" $n3 color
"violet" $n4 color
"chocolate" $n5 color
"chocolate" $n0
shape box ;
```

```
$n1 shape box ;
$n2 shape box ;
$n3 shape box ;
$n4 shape box ;
$n5 shape box;
$ns at 0.0 "$n0 label SYS0"
$ns at 0.0 "$n1 label SYS1"
$ns at 0.0 "$n2 label SYS2"
$ns at 0.0 "$n3 label SYS3"
$ns at 0.0 "$n4 label SYS4"
$ns at 0.0 "$n5 label SYS5"
set nf [open goback.nam w]
$ns namtrace-all $nf
set f [open goback.tr w]
$ns trace-all $f
$ns duplex-link $n0 $n2 1Mb 20ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns queue-limit $n0 $n2 5
$ns duplex-link $n1 $n2 1Mb 20ms DropTail
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link $n2 $n3 1Mb 20ms DropTail
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link $n3 $n4 1Mb 20ms DropTail
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link $n3 $n5 1Mb 20ms DropTail
$ns duplex-link-op $n3 $n5 orient right-down
Agent/TCP set_nam_tracevar_true
set tcp [new Agent/TCP]
$tcp set fid 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
```

```
$ftp attach-agent $tcp
$ns at 0.05 "$ftp start"
$ns at 0.06 "$tcp set windowlnit 6"
$ns at 0.06 "$tcp set maxcwnd 6"
$ns at 0.25 "$ns queue-limit $n3 $n4 0"
$ns at 0.26 "$ns queue-limit $n3 $n4 10"
$ns at 0.305 "$tcp set windowlnit 4"
$ns at 0.305 "$tcp set maxcwnd 4"
$ns at 0.368 "$ns detach-agent $n1 $tcp ; $ns detach-agent $n4 $sink"
$ns at 1.5 "finish"
$ns at 0.0 "$ns trace-annotate \"Goback N end\""
$ns at 0.05 "$ns trace-annotate \"FTP starts at 0.01\""
$ns at 0.06 "$ns trace-annotate \"Send 6Packets from SYS1 to SYS4\""
$ns at 0.26 "$ns trace-annotate \"Error Occurs for 4th packet so not sent ack for the Packet\""
$ns at 0.30 "$ns trace-annotate \"Retransmit Packet_4 to 6\""
$ns at 1.0 "$ns trace-annotate \"FTP stops\""
proc finish {} {
global ns nf
$ns flush-trace
close $nf
puts "filtering..."
#exec tclsh../bin/namfilter.tcl goback.nam
#puts "running nam..."
exec nam goback.nam &
exit 0
}$ns run
```

OUTPUT:


Applications Places System
Thu Mar 17, 11:05 AM
root
nam: goback.nam
File Views Analysis
goback.nam
play forward
0.392000
Step: 2.0ms
0
4
SYS2
2
3
SYS1
1
SYS5
5
Send 6Packets from SYS1 to SYS4
Error Occurs for 4th packet so not sent ack for the Packet
Retransmit Packet_4 to 6
new - File Browser
root@home:~/new
nam: goback.nam
Nam Console v1.13

AIM:

To Simulate and to study of Selective Repeat ARQ protocol

THEORY

Selective Repeat ARQ is a specific instance of the Automatic Repeat-reQuest (ARQ) Protocol. It may be used as a protocol for the delivery and acknowledgement of message units, or it may be used as a protocol for the delivery of subdivided message sub-units. When used as the protocol for the delivery of messages, the sending process continues to send a number of frames specified by a window size even after a frame loss. Unlike Go-Back-N ARQ, the receiving process will continue to accept and acknowledge frames sent after an initial error. The receiver process keeps track of the sequence number of the earliest frame it has not received, and sends that number with every ACK it sends. If a frame from the sender does not reach the receiver, the sender continues to send subsequent frames until it has emptied its window. The receiver continues to fill its receiving window with the subsequent frames, replying each time with an ACK containing the sequence number of the earliest missing frame. Once the sender has sent all the frames in its window, it re-sends the frame number given by the ACKs, and then continues where it left off. The size of the sending and receiving windows must be equal, and half the maximum sequence number (assuming that sequence numbers are numbered from 0 to n-1) to avoid miscommunication in all cases of packets being dropped. To understand this, consider the case when all ACKs are destroyed. If the receiving window is larger than half the maximum sequence number, some, possibly even all, of the packages that are resent after timeouts are duplicates that are not recognized as such. The sender moves its window for every packet that is acknowledged.

Advantage over Go Back N:

1. Fewer retransmissions.

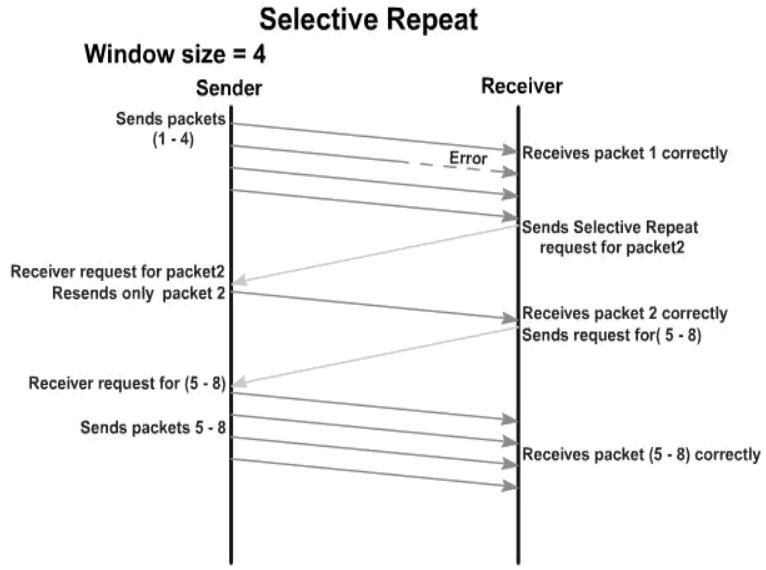
Disadvantages:

1. More complexity at sender and receiver
2. Receiver may receive frames out of sequence

ALGORITHM: SELECTIVE REPEAT

1. The source node transmits the frames continuously.
2. Each frame in the buffer has a sequence number starting from 1 and increasing up to the window size.
3. The source node has a window i.e. a buffer to store the frames. This buffer size is the number of frames to be transmitted continuously.
4. The receiver has a buffer to store the received frames. The size of the buffer depends upon the window size defined by the protocol designer.

5. The size of the window depends according to the protocol designer.
6. The source node transmits frames continuously till the window size is exhausted. If any of the frames are received with error only those frames are requested for retransmission (with a negative acknowledgement)
7. If all the frames are received without error, a cumulative positive acknowledgement is sent.
8. If there is an error in frame 3, an acknowledgement for the frame 2 is sent and then only Frame 3 is retransmitted**.** Now the window slides to get the next frames to the window.
9. If acknowledgment is transmitted with error, all the frames of window are retransmitted. Else ordinary window sliding takes place. (* In implementation part, Acknowledgment error is not considered)
10. If all the frames transmitted are errorless the next transmission is carried out for the new window.
11. This concept of repeating the transmission for the error frames only is called **Selective Repeat t**ransmission flow control protocol.



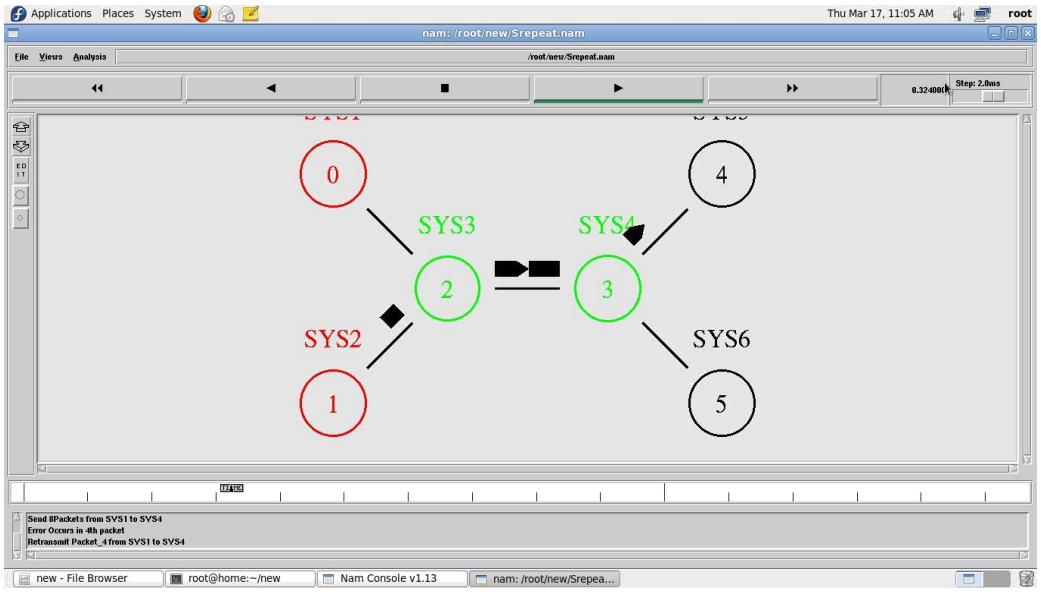
#PROGRAM FOR SELECTIVE REPEAT:

```
#send packets one by one
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n0 color "red"
$n1 color "red"
$n2 color "green"
$n3 color "green"
```

```
$n4 color "black"
$n5 color "black"
$n0 shape circle ;
$n1 shape circle ;
$n2 shape circle ;
$n3 shape circle ;
$n4 shape circle ;
$n5 shape circle ;
$ns at 0.0 "$n0 label SYS1"
$ns at 0.0 "$n1 label SYS2"
$ns at 0.0 "$n2 label SYS3"
$ns at 0.0 "$n3 label SYS4"
$ns at 0.0 "$n4 label SYS5"
$ns at 0.0 "$n5 label SYS6"
set nf [open Srepeat.nam w]
$ns namtrace-all $nf
set f [open Srepeat.tr w]
$ns trace-all $f
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link-op $n0 $n2 orient right-down
$ns queue-limit $n0 $n2 5
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link $n3 $n5 1Mb 10ms DropTail
$ns duplex-link-op $n3 $n5 orient right-down
Agent/TCP set_nam_tracevar_true
set tcp [new Agent/TCP]
$tcp set fid 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.05 "$ftp start"
$ns at 0.06 "$tcp set windowlnit 8"
$ns at 0.06 "$tcp set maxcwnd 8"
$ns at 0.25 "$ns queue-limit $n3 $n4 0"
$ns at 0.26 "$ns queue-limit $n3 $n4 10"
$ns at 0.30 "$tcp set windowlnit 1"
$ns at 0.30 "$tcp set maxcwnd 1"
$ns at 0.30 "$ns queue-limit $n3 $n4 10"
```

```
$ns at 0.47 "$ns detach-agent $n1 $tcp;$ns detach-agent $n4 $sink"
$ns at 1.75 "finish"
$ns at 0.0 "$ns trace-annotate \"Select and repeat\""
$ns at 0.05 "$ns trace-annotate \"FTP starts at 0.01\""
$ns at 0.06 "$ns trace-annotate \"Send 8Packets from SYS1 to SYS4\""
$ns at 0.26 "$ns trace-annotate \"Error Occurs in 4th packet \""
$ns at 0.30 "$ns trace-annotate \"Retransmit Packet_4 from SYS1 to SYS4\""
$ns at 1.5 "$ns trace-annotate \"FTP stops\""
proc finish {} {
global ns nf
$ns flush-trace
close $nf
puts "filtering..."
#exec tclsh../bin/namfilter.tcl srepeat.nam
#puts "running nam..."
exec nam Srepeat.nam &
exit 0
}
$ns run
```

OUTPUT:



RESULT:

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

EXNO: 10	Simulation of error correction code (like CRC)
DATE:	

AIM:

To implement error detection and error correction techniques.

THEORY:

The upper layers work on some generalized view of network architecture and are not aware of actual hardware data processing. Hence, the upper layers expect error-free transmission between the systems. Most of the applications would not function expectedly if they receive erroneous data. Applications such as voice and video may not be that affected and with some errors they may still function well. Data-link layer uses some error control mechanism to ensure that frames (data bit streams) are transmitted with certain level of accuracy. But to understand how errors is controlled, it is essential to know what types of errors may occur. CRC is a different approach to detect if the received frame contains valid data. This technique involves binary division of the data bits being sent. The divisor is generated using polynomials. The sender performs a division operation on the bits being sent and calculates the remainder. Before sending the actual bits, the sender adds the remainder at the end of the actual bits. Actual data bits plus the remainder is called a codeword. The sender transmits data bits as codewords.

ALGORITHM:

1. Open Turbo C++ software and type the program for error detection
2. Get the input in the form of bits.
3. Append 16 zeros as redundancy bits.
4. Divide the appended data using a divisor polynomial.
5. The resulting data should be transmitted to the receiver.
6. At the receiver the received data is entered.
7. The same process is repeated at the receiver.
8. If the remainder is zero there is no error otherwise there is some error in the received bits
9. Run the program.

PROGRAM

```
#include<stdio.h>
char m[50],g[50],r[50],q[50],temp[50];
void caltrans(int);
void crc(int);
void calram();
void shiftl();
 int main()
{
int n,i=0;
char ch,flag=0;
printf("Enter the frame
bits:");
while((ch=getc(stdin))
!='\n') m[i++]=ch;
n=i;
for(i=0;i<16;i++)
m[n++]='0';
m[n]='\0';
printf("Message after appending 16
zeros:%s",m); for(i=0;i<=16;i++)
g[i]='0';
g[0]=g[4]=g[11]=g[16]='1';g[17]='\0';
printf("\ngenerator:%s\n",g);
crc(n);
printf("\n\nquotient:%s",q);
caltrans(n);
printf("\ntransmitted frame:%s",m);
printf("\nEnter transmitted frame:");
scanf("\n%s",m);
printf("CRC checking\n");
crc(n);
printf("\n\nlast remainder:%s",r);
for(i=0;i<16;i++)
```

```
if(r[i]!='0')
flag=1;
else
continue;
if(flag==1)
printf("Error during transmission");
else
printf("\n\nReceived freme is correct");
}
void crc(int n)
{
int i,j;
for(i=0;i<n;i++)
temp[i]=m[i];
for(i=0;i<16;i++)
r[i]=m[i];
printf("\nintermediate remainder\n");
for(i=0;i<n-16;i++)
{
if(r[0]=='1')
{
q[i]='1';
calram();
}
else
{
q[i]='0';
shiftl();
}
r[16]=m[17+i];
r[17]='\0';
printf("\nremainder %d:%s",i+1,r);
for(j=0;j<=17;j++)
temp[j]=r[j];
```

```
}
q[n-16]='\0';
}
void calram()
{
int i,j;
for(i=1;i<=16;i++)
r[i-1]=((int)temp[i]-48)^((int)g[i]-48)+48;
}
void shiftl()
{
int i;
for(i=1;i<=16;i++)
r[i-1]=r[i];
}
void caltrans(int n)
{

int i,k=0;

for(i=n-16;i<n;i++)

m[i]=((int)m[i]-48)^((int)r[k++]-48)+48;

m[i]='\0';

}
```

OUTPUT:

Enter the Frame Bits:

1011

The msg after appending 16 zeros:

10110000000000000000

The Transmitted frame is:**10111011000101101011**

Enter the transmitted Frame

10111011000101101011

Received msg:**10111011000101101011**

The Remainder is:**0000000000000000**

Received frame is correct.

RESULT:

Thus the error detection and error correction is implemented successfully.