PRATHYUSHA ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

for

CS3591-COMPUTER NETWORKS LABORATORY

(Regulation 2021, V Semester)

(Odd Semester)

ACADEMIC YEAR: 2023 – 2024

PREPARED BY

THAMBA MESHACH W,

Assistant Professor / CSE

CS3591 COMPUTER NETWORKS LABORATORY L T P C
3 0 2 4

COURSE OBJECTIVES:

- To understand the concept of layering in networks.
- To know the functions of protocols of each layer of TCP/IP protocol suite.
- To visualize the end-to-end flow of information.
- To learn the functions of network layer and the various routing protocols
- To familiarize the functions and protocols of the Transport layer

LIST OF EXPERIMENTS:

1. Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and trace route PDUs using a network protocol analyzer and examine.
2. Write a HTTP web client program to download a web page using TCP sockets.
3. Applications using TCP sockets like: a) Echo client and echo server b) Chat
4. Simulation of DNS using UDP sockets.
5. Use a tool like Wireshark to capture packets and examine the packets
6. Write a code simulating ARP /RARP protocols.
7. Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.
8. Study of TCP/UDP performance using Simulation tool.
9. Simulation of Distance Vector/ Link State Routing algorithm.
10. Simulation of an error correction code (like CRC)

TOTAL: 30 PERIODS

COURSE OUTCOMES:

At the end of this course, the students will be able to:
CO 1: Explain the basic layers and its functions in computer networks.
CO 2: Understand the basics of how data flows from one node to another.
CO 3: Analyze routing algorithms.
CO 4: Describe protocols for various functions in the network.
CO 5: Analyze the working of various application layer protocols.

Ex No : 1 **Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute.**

Date: **Capture ping and traceroute PDUs using a network protocol analyzer and examine.**

AIM: To Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute ping.

ALGORITHM - Commands:

1. **Tcpdump:**

   - **Display traffic between 2 hosts:**

     To display all traffic between two hosts (represented by variables host1 and host2):

     # tcpdump host host1 and host2

   - **Display traffic from a source or destination host only:**

     To display traffic from only a source (src) or destination (dst) host:

     #tcpdump src host

     #tcpdump dst host

   - **Display traffic for a specific protocol**

     Provide the protocol as an argument to display only traffic for a specific protocol,

     for example tcp, udp, icmp, arp

     # tcpdump protocol

     For example to display traffic only for the tcp traffic :

     # tcpdump tcp

   - **Filtering based on source or destination port**

     To filter based on a source or destination port:

     # tcpdump src port ftp

     # tcpdump dst port http

2. **Netstat**

   Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX and other operating systems.

   #netstat

3. **Ipconfig**

From the command prompt, type **ipconfig** to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapter.

#ipconfig

4. **nslookup**

The **nslookup** (which stands for *name server lookup*) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System.

#nslookup

5. **Trace route:**

With the tracert command shown above, we're asking tracert to show us the path from the local computer all the way to the network device with the hostname

www.google.com.

#tracert google.com

6. **Ping:**

The ping command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not.

ping172.16.6.2

OUTPUT : Commands

1) **Tcpdump:**

- **Display traffic between 2 hosts:**

To display all traffic between two hosts (represented by variables host1 and host2):

tcpdump host host1 and host2

- **Display traffic from a source or destination host only:**

To display traffic from only a source (src) or destination (dst) host:

#tcpdumpsrc host

#tcpdumpdst host

❖ **Display traffic for a specific protocol**

Provide the protocol as an argument to display only traffic for a specific protocol, for example tcp, udp, icmp, arp

 # tcpdump protocol

For example to display traffic only for the tcptraffic :

 # tcpdumptcp

❖ **Filtering based on source or destination port**

To filter based on a source or destination port:

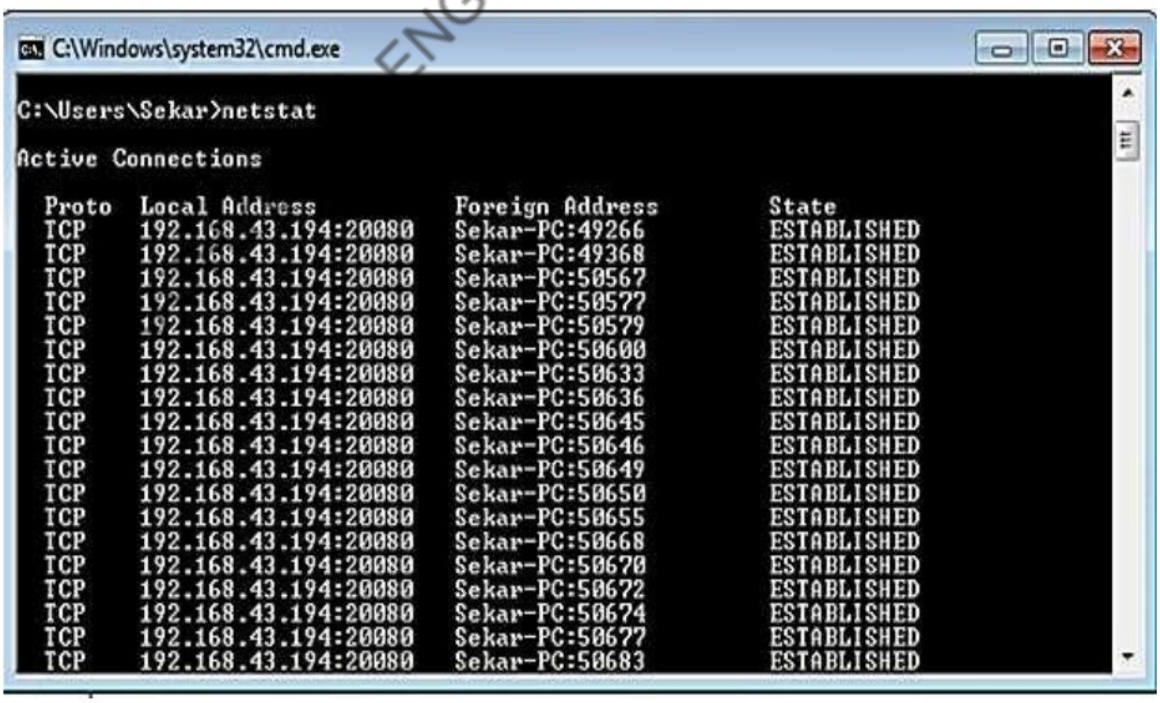
 # tcpdumpsrc port ftp

 # tcpdumpdst port http

2) Netstat:

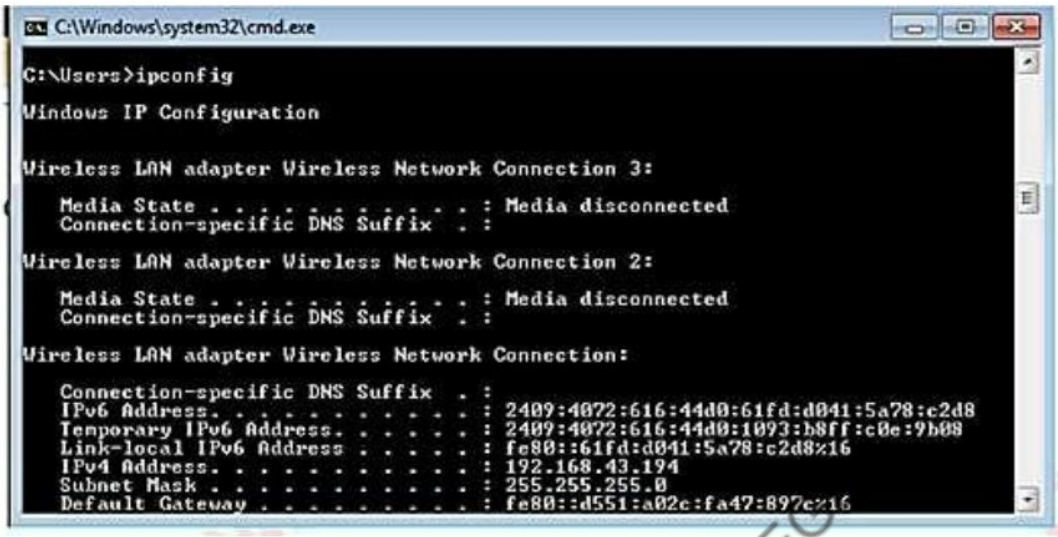
Displays protocol statistics and current TCP/IP network connections.

>netstat



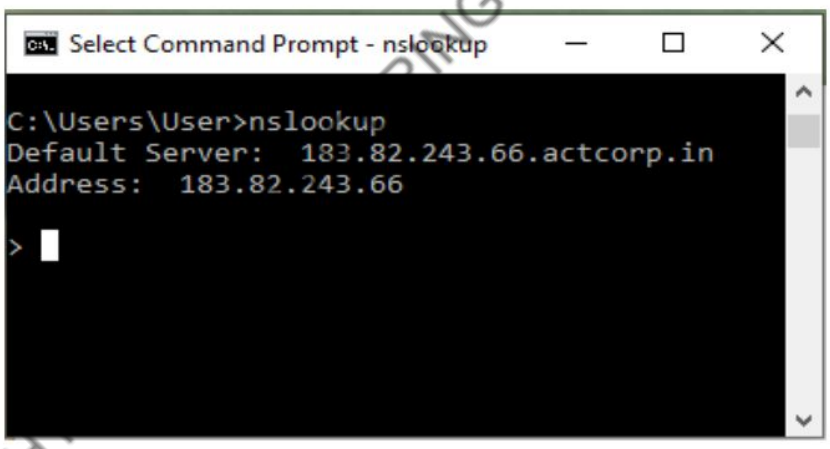
3) Ipconfig:

>ipconfig



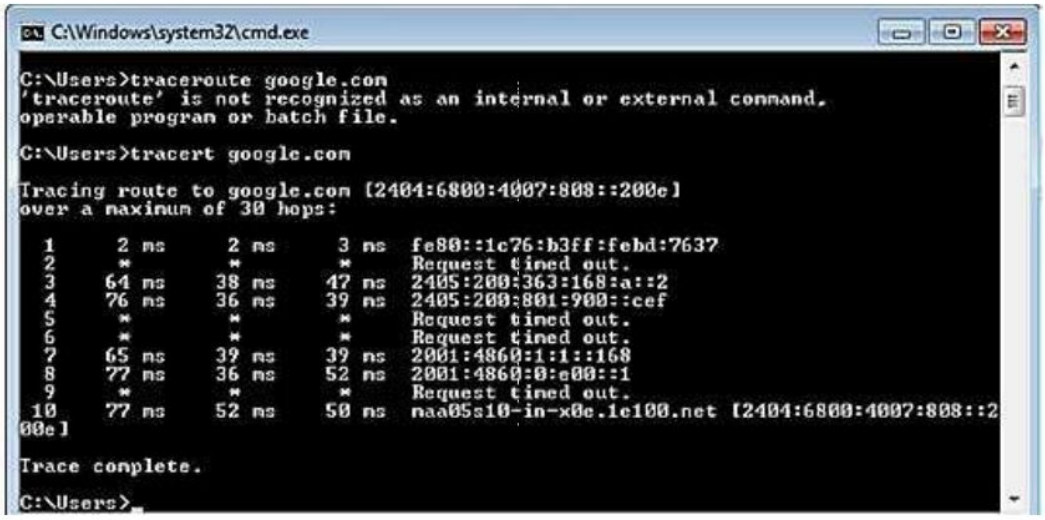
4) nslookup

>nslookup



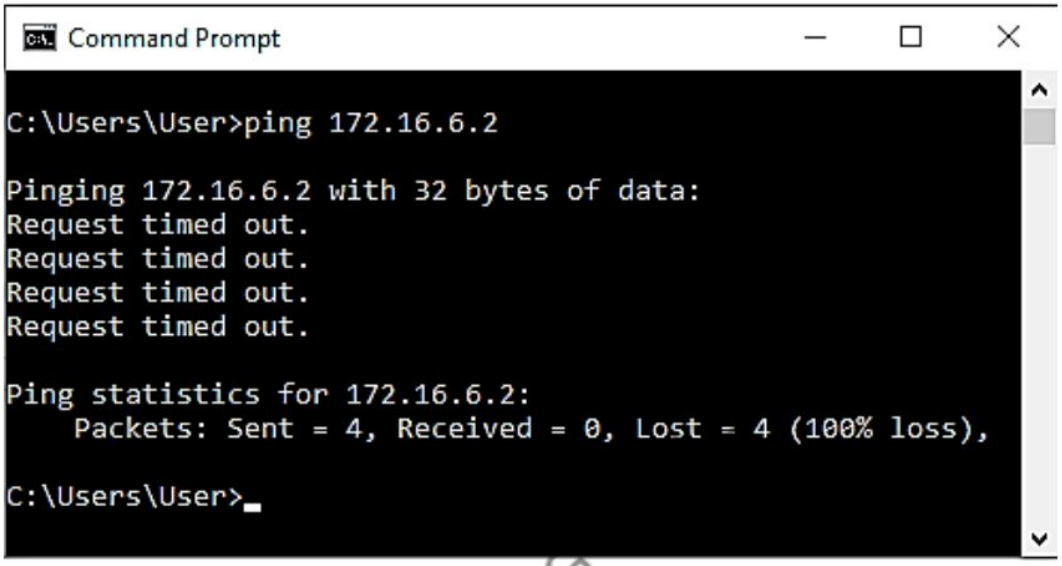
5) Trace route:

>tracert google.com



6) **Ping:**

>ping172.16.6.2



RESULT:

Thus the various networks commands like tcpdump, netstat, ifconfig, nslookup and traceroute ping are executed successfully.

Ex No: 2 **Write a HTTP web client program to download a web page using TCP sockets**

Date:

AIM:

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

ALGORITHM:

Client:

1. Start.
2. Create socket and establish the connection with the server.
3. Read the image to be uploaded from the disk
4. Send the image read to the server
5. Terminate the connection
6. Stop.

Server:

1. Start
2. Create socket, bind IP address and port number with the created socket and make server a listening server.
3. Accept the connection request from the client
4. Receive the image sent by the client.
5. Display the image.
6. Close the connection.
7. Stop.

PROGRAM

Client

```
import javax.swing.*;
import java.net.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.awt.image.BufferedImage; import
java.io.ByteArrayOutputStream; import
java.io.File;
import java.io.IOException; import
javax.imageio.ImageIO;
public class Client
{
public static void main(String args[]) throws Exception
{
Socket soc;
BufferedImage img = null;
soc=new
Socket("localhost",4000);
System.out.println("Client is running.");
try
{
System.out.println("Reading image from disk. ");
img = ImageIO.read(new File("digital_image_processing.jpg")); ByteArrayOutputStream baos = new
ByteArrayOutputStream();
ImageIO.write(img, "jpg", baos);
baos.flush();
byte[]  bytes = baos.toByteArray();
baos.close();
System.out.println("Sending image to server.");
OutputStream out = soc.getOutputStream();
```

```
DataOutputStream dos = new DataOutputStream(out);
dos.writeInt(bytes.length);
dos.write(bytes, 0, bytes.length);
System.out.println("Image sent to server. ");
dos.close();
out.close();
}
catch (Exception e)
{
System.out.println("Exception: " + e.getMessage());
soc.close();
}
soc.close();
}
}
```

Server

```
import java.net.*;
import java.io.*;
import java.awt.image.*;
importjavax.imageio.*;
importjavax.swing.*;
class Server
{
 public static void main(String args[]) throws Exception
{
ServerSocket server=null;
Socket socket;
server=new ServerSocket(4000);
System.out.println("Server Waiting for image");
socket=server.accept();
System.out.println("Client connected.");
InputStream in = socket.getInputStream();
```

```
DataInputStream dis = new DataInputStream(in);
Int len = dis.readInt();
System.out.println("Image Size: " + len/1024 + "KB");
byte[] data = new byte[len];
dis.readFully(data);
dis.close();
in.close();
InputStream ian = new ByteArrayInputStream(data);
BufferedImage bImage = ImageIO.read(ian);
JFrame f = new JFrame("Server");
ImageIcon icon = new ImageIcon(bImage);
JLabel l = new JLabel();
l.setIcon(icon);
f.add(l);
f.pack();
f.setVisible(true);
}
}
```

OUTPUT:

When you run the client code, following output screen would appear on client side.

RESULT:

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

Ex No: 3 Applications using TCP sockets like: Echo client and echo server, Chat

Date:

AIM

To write a java program for application using TCP Sockets Links

A) Echo client and echo server

ALGORITHM

Client

1. Start
2. Create the TCP socket
3. Establish connection with the server
4. Get the message to be echoed from the user
5. Send the message to the server
6. Receive the message echoed by the server
7. Display the message received from the server
8. Terminate the connection
9. Stop

Server

1. Start
2. Create TCP socket, make it a listening socket
3. Accept the connection request sent by the client for connection establishment
4. Receive the message sent by the client
5. Display the received message
6. Send the received message to the client from which it receives
7. Close the connection when client initiates termination and server becomes a listening server, waiting for clients.
8. Stop.

PROGRAM:

EServer.java

```
import java.net.*;
import java.io.*;
public class EServer
{
public static void main(String args[])
{
  ServerSocket s=null;
  String line;
  DataInputStream is;
  PrintStream ps;
  Socket c=null;
  try
  {
  s=new ServerSocket(9000);
  }
  catch(IOException e)
  {
  System.out.println(e);
  }
  try
  {
  c=s.accept();
  is=new DataInputStream(c.getInputStream());
  ps=new PrintStream(c.getOutputStream());
  while(true)
  {
  line=is.readLine();
  ps.println(line);
  }
```

```
}
catch(IOException e)
{
System.out.println(e);
}
}
}
```

EClient.java

```
import java.net.*;
import java.io.*;
public class EClient
{
 public static void main(String arg[])
{
Socket c=null; String line;
DataInputStream is,is1;
PrintStream os;
try
{
InetAddress ia = InetAddress.getLocalHost();
c=new Socket(ia,9000);
}
catch(IOException e)
{
System.out.println(e);
}
try
{
os=new PrintStream(c.getOutputStream());
 is=new DataInputStream(System.in);
is1=new DataInputStream(c.getInputStream());
```

```
while(true)
{
System.out.println("Client:");
line=is.readLine();
os.println(line);
System.out.println("Server:" + is1.readLine());
}
}
catch(IOException e)
{
 System.out.println("Socket Closed!");
}
}
}
```

OUTPUT

Server

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

Client

```
C:\Program Files\Java\jdk1.5.0\bin>javac EClient.java
C:\Program Files\Java\jdk1.5.0\bin>java EClient
Client:  Hai Server
Server:Hai Server
Client: Hello
Server:Hello
Client:end
Server:end
Client:ds
Socket Closed!
```

B.Chat

ALGORITHM

Client

1. Start
2. Create the UDP datagram socket
3. Get the request message to be sent from the user
4. Send the request message to the server
5. If the request message is “END” go to step 10
6. Wait for the reply message from the server
7. Receive the reply message sent by the server
8. Display the reply message received from the server
9. Repeat the steps from 3 to 8
10. Stop

Server

1. Start
2. Create UDP datagram socket, make it a listening socket
3. Receive the request message sent by the client
4. If the received message is “END” go to step 10
5. Retrieve the client’s IP address from the request message received
6. Display the received message
7. Get the reply message from the user
8. Send the reply message to the client
9. Repeat the steps from 3 to 8.
10. Stop.

PROGRAM

UDPserver.java

```
import java.io.*;
import java.net.*;
class UDPserver
{
public static DatagramSocket ds;
public static byte buffer[]=new byte[1024];
public static int clientport=789,serverport=790;
public static void main(String args[])throws Exception
{
ds=new DatagramSocket(clientport);
System.out.println("press ctrl+c to quit the program");
BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));
InetAddress ia=InetAddress.geyLocalHost();
 while(true)
{
DatagramPacket p=new  DatagramPacket(buffer,buffer.length);
ds.receive(p);
String psx=new String(p.getData(),0,p.getLength());
System.out.println("Client:" + psx);
System.out.println("Server:");
String str=dis.readLine();
if(str.equals("end"))
break;
buffer=str.getBytes();
ds.send(new DatagramPacket(buffer,str.length(),ia,serverport));
}
}
}
```

UDPclient.java

```
import java .io.*;
import java.net.*;
class UDPclient
{
public static DatagramSocket ds;
public static int clientport=789,serverport=790;
public static void main(String args[])throws Exception
{
byte buffer[]=new byte[1024];
ds=new DatagramSocket(serverport);
BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));
System.out.println("server waiting");
InetAddress ia=InetAddress.getLocalHost();
while(true)
{
System.out.println("Client:");
String str=dis.readLine();
if(str.equals("end"))
break;
buffer=str.getBytes();
ds.send(new DatagramPacket(buffer,str.length(),ia,clientport));
DatagramPacket p=new DatagramPacket(buffer,buffer.length); ds.receive(p);
String psx=new String(p.getData(),0,p.getLength());
System.out.println("Server:" + psx);
}
}
}
```

OUTPUT:

Server:

C:\Program Files\Java\jdk1.5.0\bin>javac UDPserver.java

C:\Program Files\Java\jdk1.5.0\bin>java UDPserver

pressctrl+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:HelloClie

Client:How are You

Server:I am Fine

Client:end

RESULT:

Thus the java application program using TCP Sockets was developed and executed successfully.

Ex No: 4 **Simulation of DNS using UDP Sockets**

Date:

AIM

To write a java program for DNS application

ALGORITHM

Server

1. Start
2. Create UDP datagram socket
3. Create a table that maps host name and IP address
4. Receive the host name from the client
5. Retrieve the client's IP address from the received datagram
6. Get the IP address mapped for the host name from the table.
7. Display the host name and corresponding IP address
8. Send the IP address for the requested host name to the client
9. Stop.

Client

1. Start
2. Create UDP datagram socket.
3. Get the host name from the client
4. Send the host name to the server
5. Wait for the reply from the server
6. Receive the reply datagram and read the IP address for the requested host name
7. Display the IP address.
8. Stop.

PROGRAM

Udpdnsserver.java

```
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)
{
DatagramSocketserver socket=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();
}
}
}
```

udpdnsclient.java

```
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));
DatagramSocketclient socket = 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 java application program using UDP Sockets to implement DNS was developed and executed successfully

Ex No:5 **Use a tool like Wireshark to capture packets and examine the packets**

Date:

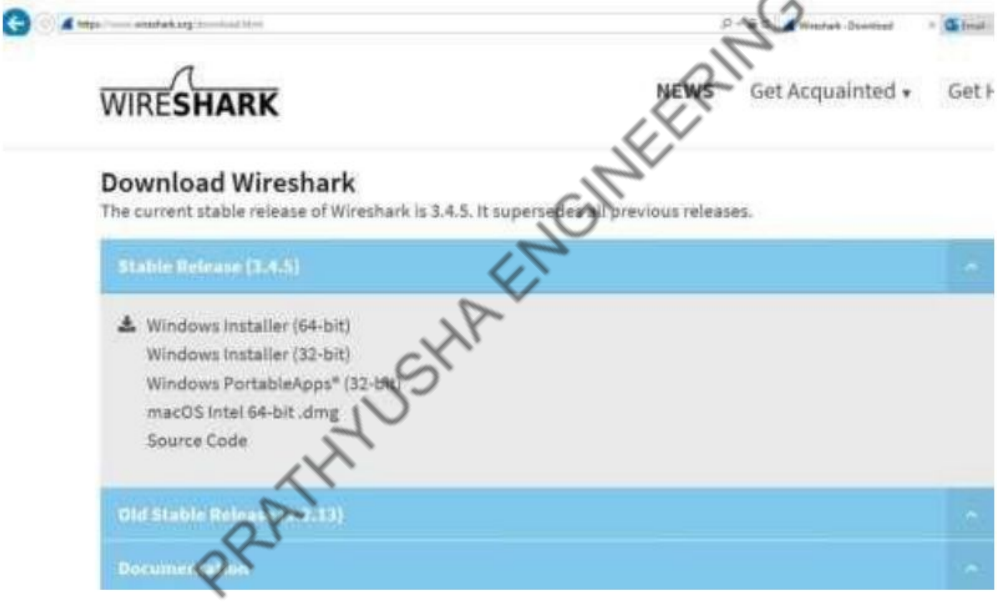
AIM:

To use a tool like Wireshark to capture packets and examine the packets

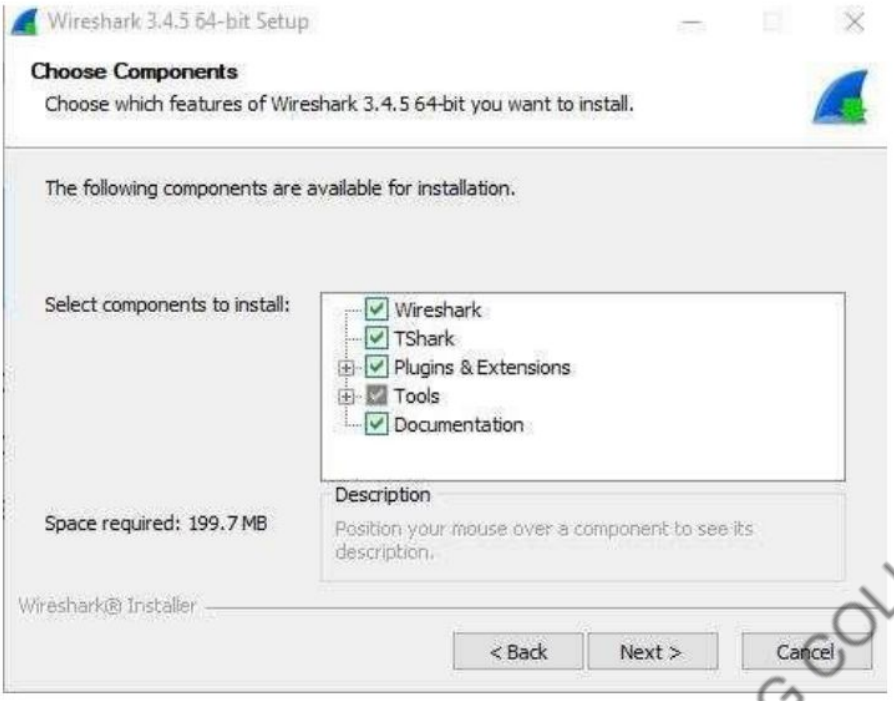
STEPS:

1. Installing Wireshark

- Go to https://www.wireshark.org/download.html to visit the Wireshark download page.
  Ensure that the full URL is used.
  The output will look similar to the following:



- Download the Windows Installer (64 bit).
  Click on the link for version of software, and see a pop-up box at the foot of the screen.
  Choose run, and when the UAC prompt appears, choose Yes.
  This will bring up the installation wizard.

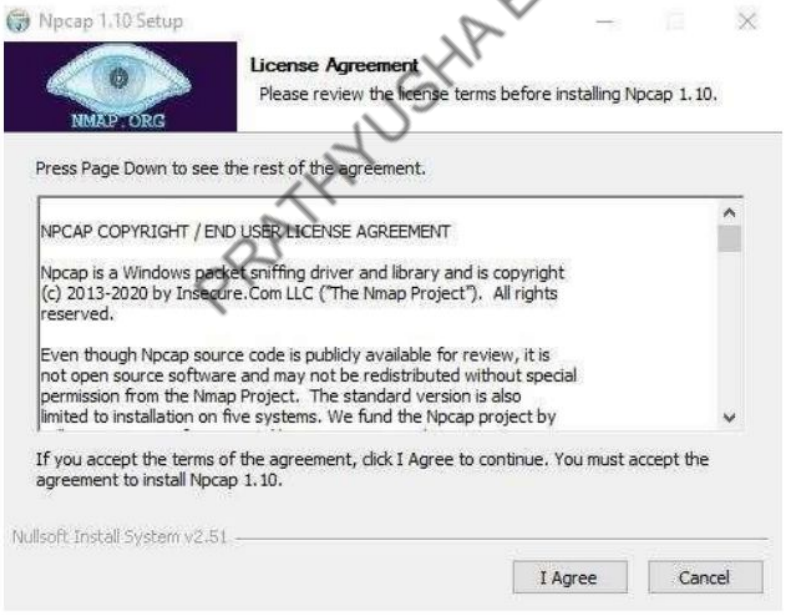


- Keep pressing 'Next' and accept the defaults.

  The installation will commence, and the pop-up box below will appear.

  Accept the license agreement and press I Agree.

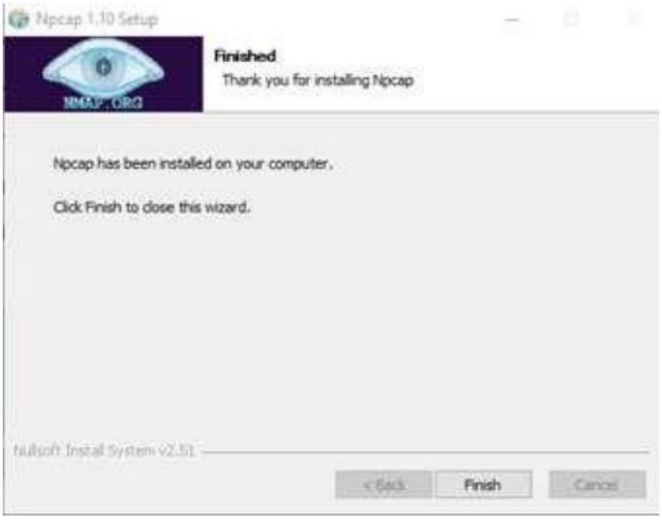
  Accept the default settings.



- Accept the default settings by pressing Next.

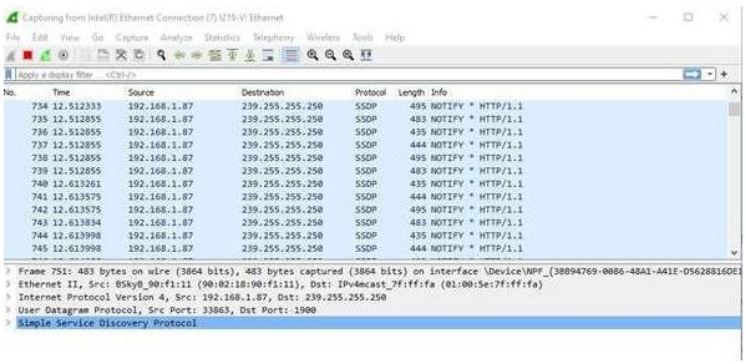
  The following wizard will appear.

  Press Finish.



- The Wireshark installation will still be running in the background.
  It should take roughly another 2-3 minutes.
  The wizard will appear to say the installation is complete.
  Select Next, then Finish.
  You will now see a Wireshark shortcut on the desktop, the same as below:

- Double-click it and choose your network interface.
  When your Wireshark console appears, it should look similar to that shown below.
  If you need to change the interface, go to Capture and select Options.



- The view above shows The Main Window, which is broken into different sections:

  The Menu: This is broken into the following 11 headings:

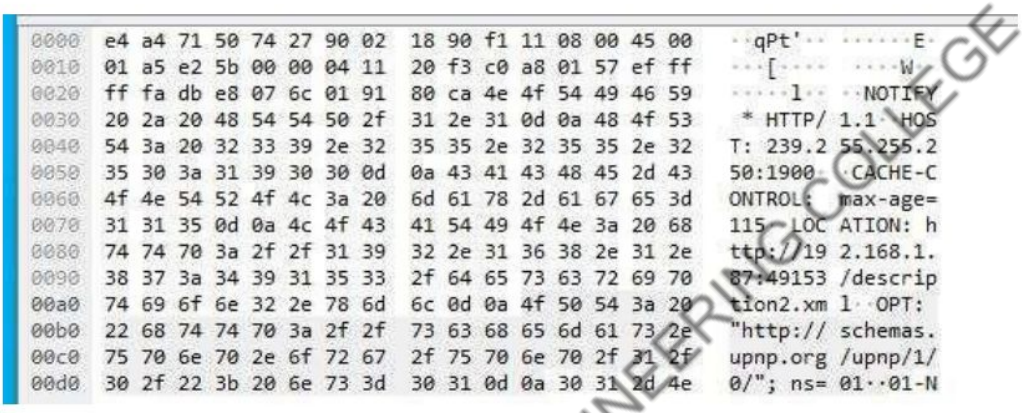
  File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, Help.

  The Filter Toolbar: This has a filter pane when you type in the protocol that you want to view.

  The Packet List: This shows all packets that are captured and is shown in blue in the preceding image.

  The Packet Details Pane: This is the gray area that shows the protocol fields of the packet.

  The Packet Bytes Pane: This shows a canonical hex dump of the packet data.



2. Capturing Packets

- Go to the Capture drop-down menu option.

  Options lets you change the network interface.

  Press the shark symbol with the word Start.

  Once you are finished capturing the traffic, press the red square with the word Stop on it.

  These menus are context-sensitive; for example, the Stop button does not become live until after the Start button appears.

- Always have web browser ready before pressing 'Start'.

  Once start up Wireshark, capture a vast amount of traffic.

  After capturing it, filter the different types of traffic.

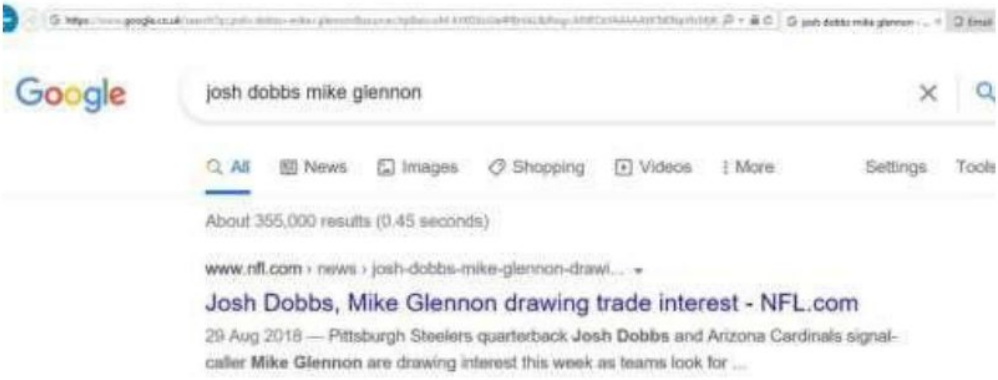
3. Preparation Before Capture

- Go to Google and search for 'Josh Dobbs Mike Glennon'.

  The top of the search list should be similar to that shown below.

  If that article is not available, go to the Amazon website instead and search for the Ian Neil Security+.

  The following instructions will be the same.



4. Starting the Capture

- Start Wireshark, go to Capture, then press Start.

  Go to Google and press the hyperlink for the preceding article.

  Massive number of packets being captured.

5. Stop the Capture

- Go to Wireshark, Capture menu, and Press Stop.

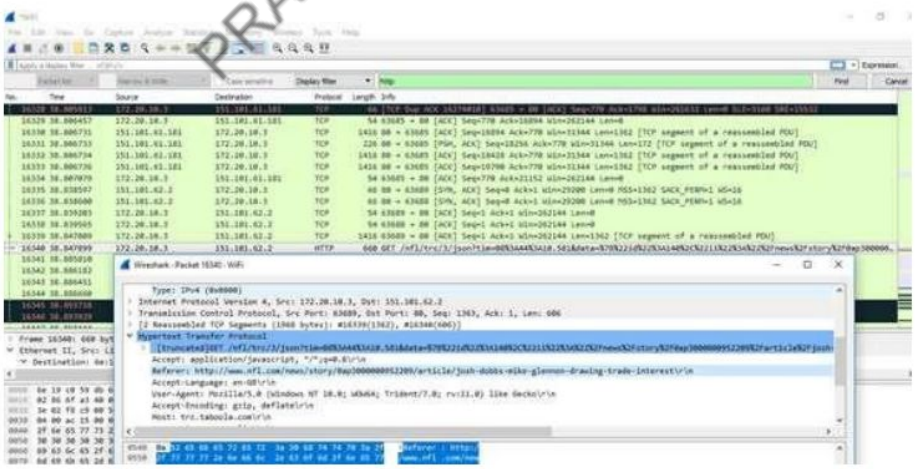
  Captured quite a few packets.

  On this occasion, capture over 20,000 packets in about 3 minutes.

6. Saving the Capture File

- On the Wireshark console, in the top left-hand corner, choose File.

  Then select ‘Save as‘ and save it as a pcap file (a packet capture file).



7. Filtering the Capture File (cap)

- In the captured packet, insert the filter http.

  Now only TCP and HTTP traffic can been viewed.

  The packet 16340 relates to arrival at the articles on the nfl.com website.

  The IP Address is 172.20.10.1, and the destination is 151.101.62.2.

  The request to go to a website uses the HTTP verb GET.

  Now search the trace for the packet when arrived at this article?

  Open the frame in the packet details pane.

  It is using IPV4, and the traffic is TCP.

  Expand the HTTP packet, and the referrer will show the page that isited.

RESULT:

Thus Wireshark tool is used to capture packets and examine the packets

Ex No:6 **Write a code simulating ARP /RARP protocols**

Date:

A) Program for Address Resolution Protocol (ARP) using TCP

AIM:

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

ALGORITHM:

Client

1. Start the program
2. Create socket and establish connection with the server.
3. Get the IP address to be converted into MAC address from the user.
4. Send this IP address to server.
5. Receive the MAC address for the IP address from the server.
6. Display the received MAC address
7. Terminate the connection

Server

1. Start the program
2. Create the socket, bind the socket created with IP address and port number and make it a listening socket.
3. Accept the connection request when it is requested by the client.
4. Server maintains the table in which IP and corresponding MAC addresses are stored.
5. Receive the IP address sent by the client.
6. Retrieve the corresponding MAC address for the IP address and send it to the client.
7. Close the connection with the client and now the server becomes a listening server waiting for the connection request from other clients
8. Stop

A) PROGRAM FOR ARP

Clientarp.java:

```
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);
}
}
}
```

Serverarp.java:

```
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(inti=0;i<ip.length;i++)
{
if(str.equals(ip[i]))
{
dout.writeBytes(mac[i]+'\n');
break;
}
}
obj.close();
}
}
```

```
catch(Exception e)
{
System.out.println(e);
}
}
}
```

Output:

```
E:\networks>java Serverarp
E:\networks>java Clientarp
Enter the Logical address(IP):165.165.80.80
The Physical Address is: 6A:08:AA:C2
```

B) Program for Reverse Address Resolution Protocol (RARP) using TCP

AIM:

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

ALGORITHM:

Client

1. Start the program
2. Create datagram socket
3. Get the MAC address to be converted into IP address from the user.
4. Send this MAC address to server using UDP datagram.
5. Receive the datagram from the server and display the corresponding IP address.
6. Stop

Server

1. Start the program.
2. Server maintains the table in which IP and corresponding MAC addresses are stored.
3. Create the datagram socket
4. Receive the datagram sent by the client and read the MAC address sent.
5. Retrieve the IP address for the received MAC address from the table.
6. Display the corresponding IP address.
7. Stop

PROGRAM FOR RARP

Clientrarp12.java:

```
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);
}
}
}
```

Serverrarp12.java:

```
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(inti=0;i<ip.length;i++)
{
if(s.equals(mac[i]))
```

```
{
sendbyte=ip[i].getBytes();
DatagramPacket sender = new DatagramPacket( sendbyte,sendbyte. length,addr,port);
server.send(sender);
break;
}
}
break;
}
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```


Output:

```
I:\ex>java Serverrarp12
I:\ex>java Clientrarp12
Enter the Physical address (MAC): 6A:08:AA:C2
The Logical Address is(IP): 165.165.80.80
```

RESULT :

Thus the program for implementing to display simulating ARP and RARP protocols was executed successfully and output is verified.

Ex No: 7 **Study of Network simulator (NS) and Simulation of Congestion Control**

Date: **Algorithms using NS**

AIM:

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

Procedure: NETWORK SIMULATOR (NS2)

Ns Overview

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

Ns functionalities

- Routing, Transportation,
- Traffic sources,
- Queuing disciplines,
- QoS

Congestion Control Algorithms

- Slow start
- Additive increase/multiplicative decrease
- Fast retransmit and Fast recovery

Case Study: A simple Wireless network.

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

PROGRAM

```
include<wifi_lte/wifi_lte_rtable.h>
structr_hist_entry *elm, *elm2;
int num_later = 1;
elm = STAILQ_FIRST(&r_hist_);
while (elm != NULL &&num_later<= num_dup_acks_)
{
num_later;
elm = STAILQ_NEXT(elm, linfo_);
}
if (elm != NULL)
{
elm = findDataPacket InRecvHistory(STAILQ_NEXT(elm,linfo_));
if (elm != NULL)
{
elm2 = STAILQ_NEXT(elm, linfo_);
while(elm2 != NULL)
{
if (elm2->seq_num_ <seq_num&& elm2->t_recv_ < time)
{
STAILQ_REMOVE(&r_hist_,elm2,r_his t_entry,linfo_);
delete elm2;
}
else
elm = elm2;
elm2 = STAILQ_NEXT(elm, linfo_);
}
}
}
}
Void DCCPTFRCAgent::removeAcksRecvHistory()
```

```
{
structr_hist_entry *elm1 = STAILQ_FIRST(&r_hist_);
structr_hist_entry *elm2;
int num_later = 1;
while (elm1 != NULL &&num_later<= num_dup_acks_)
{
num_later;
elm1 = STAILQ_NEXT(elm1, linfo_);
}
if(elm1 == NULL)
return;
elm2 = STAILQ_NEXT(elm1, linfo_);
while(elm2 != NULL)
{
if (elm2->type_ == DCCP_ACK){
STAILQ_REMOVE(&r_hist_,elm2,r_hist_entry,linfo_);
delete elm2;
}
Else
{
elm1 = elm2;
}
elm2 = STAILQ_NEXT(elm1, linfo_);
}
}
inliner_hist_entry *DCCPTFRCAgent:: findDataPacketInRecvHistory(r_hist_entry *start)
{
while(start != NULL && start->type_ == DCCP_ACK)
start = STAILQ_NEXT(start,linfo_);
return start;
}
```

Result:

Thus we have Studied Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.

Ex No: 8 **Study of TCP/UDP performance using Simulation tool.**

Date:

AIM:

To simulate the performance of TCP/UDP using NS2.

TCP Performance – Algorithm

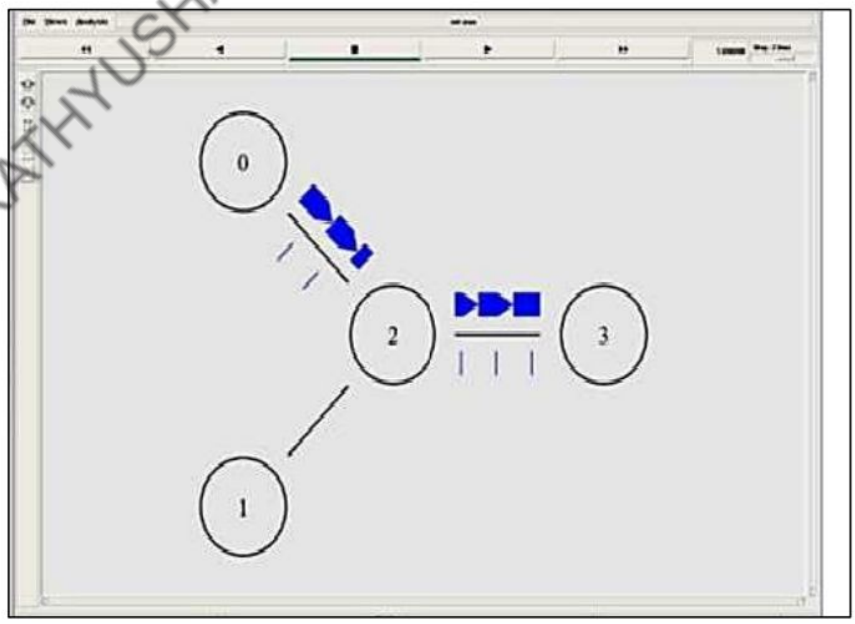
1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.
7. Create the applications and attach them to the tcp agent.
8. Connect tcp and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

PROGRAM

```
set ns [new Simulator]
$ns color 0 Blue
$ns color 1 Red
$ns color 2 Yellow
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set f [open tcpout.tr w]
$ns trace-all $f
setnf [open tcpout.nam w]
$ns namtrace-all $nf
$ns duplex-link $n0 $n2 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
```

```
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n2 $n3 queuePos 0.5
settcp [new Agent/TCP]
$tcp set class_ 1
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $tcp
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.2 "$ftp start"
$ns at 1.35 "$ns detach-agent $n1 $tcp ; $ns detach-agent $n3 $sink"
$ns at 3.0 "finish"
$ns flush-trace
close $f
close $nf
puts "Running nam.."
execxgraph tcpout.tr -geometry 600x800 &
execnamtcpout.nam&exit 0
}
```

OUTPUT:



UDP Performance - Algorithm :

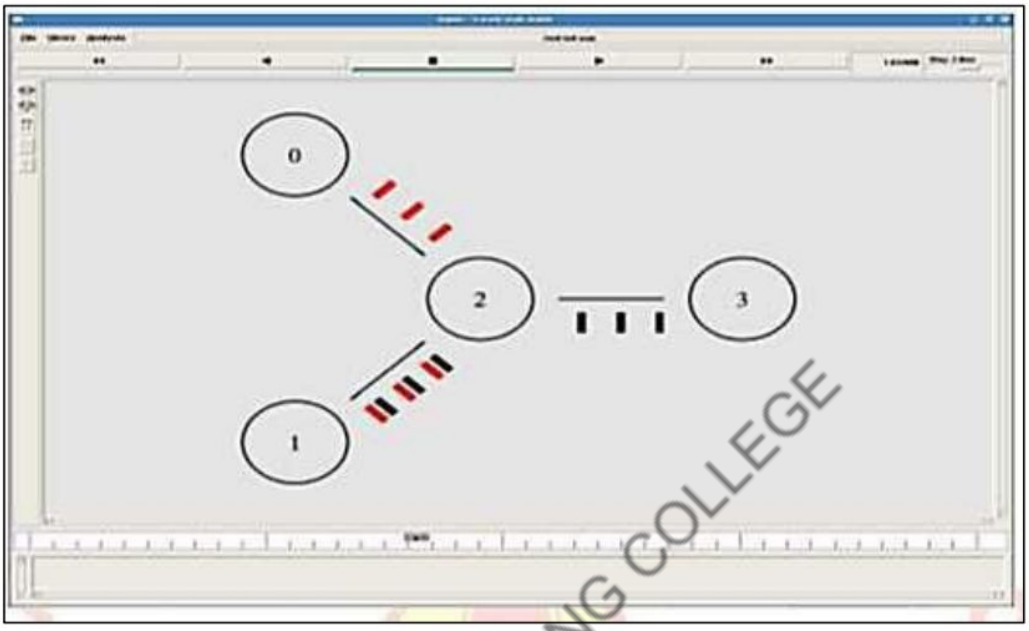
1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.
7. Create the applications and attach them to the UDP agent.
8. Connect udp and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

PROGRAM

```
set ns [new Simulator]
$ns color 0 Blue
$ns color 1 Red
$ns color 2 Yellow
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set f [open udpout.tr w]
$ns trace-all $f
setnf [open udpout.nam w]
$ns namtrace-all $nf
$ns duplex-link $n0 $n2 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n1 $n2 orient right-down
$ns duplex-link-op $n2 $n3 orient right
$ns duplex-link-op $n2 $n3 queuePos 0.5
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
set udp1 [new Agent/UDP]
$ns attach-agent $n3 $udp1
```

```
$udp1 set class_ 0
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n1 $null0
set null1 [new Agent/Null]
$ns attach-agent $n1 $null1
$ns connect $udp0 $null0
$ns connect $udp1 $null1
$ns at 1.0 "$cbr0 start"
$ns at 1.1 "$cbr1 start"
puts [$cbr0 set packetSize_]
puts [$cbr0 set interval_]
$ns at 3.0 "finish"
proc finish {} {
global ns f nf
$ns flush-trace
close $f
close $nf
puts "Running nam.."
execnamudpout.nam&
exit 0
}
$ns run
```

OUTPUT:



RESULT

Thus, the study of TCP/UDP performance is done successfully.

Ex No: 9 Simulation of Distance Vector/ Link State Routing algorithm.

Date:

A) LINK STATE ROUTING ALGORITHM

AIM:

To simulate the Link state routing protocols using NS2.

ALGORITHM:

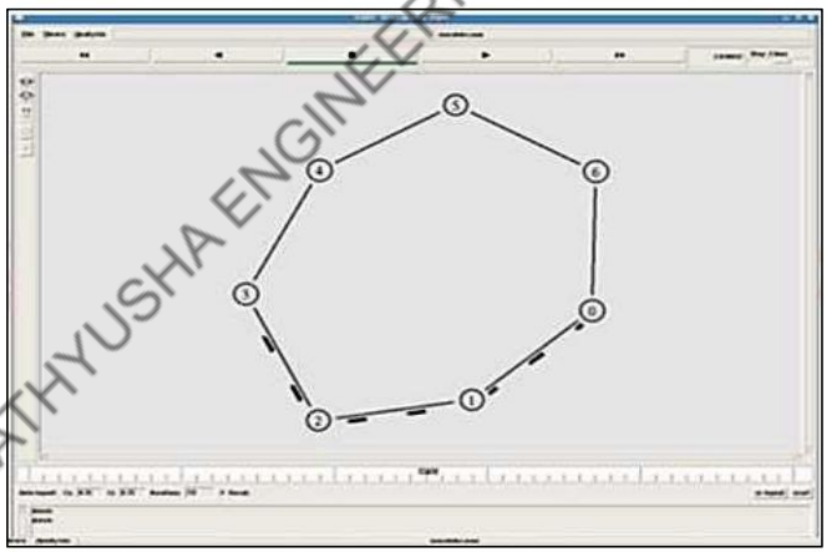
1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.
7. Create the applications and attach them to the udp agent.
8. Connect udp and null..
9. At 1 sec the link between node 1 and 2 is broken.
10. At 2 sec the link is up again.
11. Run the simulation.

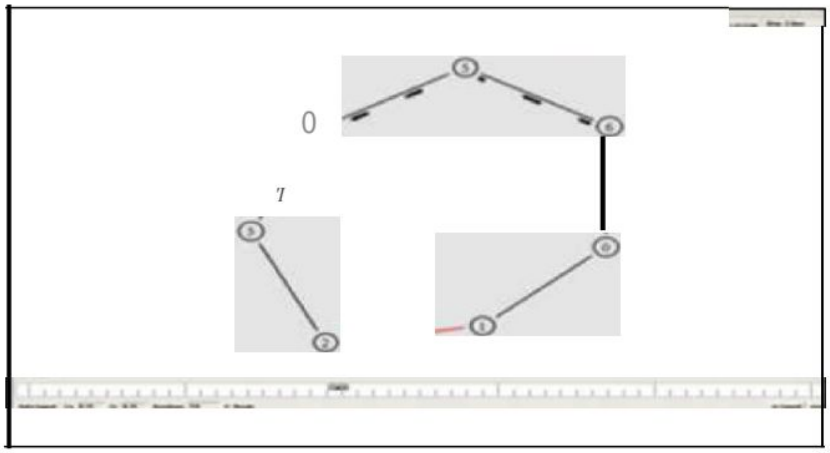
PROGRAM

```
set ns [new Simulator]
$ns rtproto LS
setnf [open linkstate.nam w]
$ns namtrace-all $nf
set f0 [open linkstate.tr w]
$ns trace-all $f0
proc finish {}
{
global ns f0 nf
$ns flush-trace
close $f0
close $nf
exec namlinkstate.nam&
exit 0
}
for {set i 0} {$i<7} {incri}
{
set n($i) [$ns node]
}
for {set i 0} {$i<7} {incri}
{
$ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
```

```
$ns attach-agent $n(3) $null0
$ns connect $udp0 $null0
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

OUTPUT:





B) DISTANCE VECTOR ROUTING ALGORITHM

AIM:

To simulate the Distance vector routing protocols using NS2.

ALGORITHM:

1. Create a simulator object
2. Set routing protocol to Distance Vector routing
3. Trace packets on all links onto NAM trace and text trace file
4. Define finish procedure to close files, flush tracing and run NAM
5. Create eight nodes
6. Specify the link characteristics between nodes
7. Describe their layout topology as a octagon
8. Add UDP agent for node n1
9. Create CBR traffic on top of UDP and set traffic parameters.
10. Add a sink agent to node n4
11. Connect source and the sink
12. Schedule events as follows:
    a. Start traffic flow at 0.5
    b. Down the link n3-n4 at 1.0
    c. Up the link n3-n4 at 2.0
    d. Stop traffic at 3.0
    e. Call finish procedure at 5.0
13. Start the scheduler
14. Observe the traffic route when link is up and down
15. View the simulated events and trace file analyze it
16. Stop

PROGRAM

```
#Distance vector routing protocol – distvect.tcl
#Create a simulator object
set ns [new Simulator]
#Use distance vector routing
$ns rtproto DV
#Open the nam trace file
setnf [open out.nam w]
$ns namtrace-all $nf
# Open tracefile
setnt [open trace.tr w]
$ns trace-all $nt
#Define 'finish' procedure
proc finish {}
{
global ns nf
$ns flush-trace
#Close the trace file
close $nf
#Executenam on the trace file
execnam -a out.nam&
exit 0
}
# Create 8 nodes
# set n1 [$ns ode]
# set n2 [$ns ode]
# set n3 [$ns ode]
# set n4 [$ns ode]
# set n5 [$ns ode]
# set n6 [$ns ode]
# set n7 [$ns ode]
# set n8 [$ns ode]
```

```
# Specify link characterestics
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns duplex-link $n5 $n6 1Mb 10ms DropTail
$ns duplex-link $n6 $n7 1Mb 10ms DropTail
$ns duplex-link $n7 $n8 1Mb 10ms DropTail
$ns duplex-link $n8 $n1 1Mb 10ms DropTail
# specify layout as a octagon
$ns duplex-link-op $n1 $n2 orient left-up
$ns duplex-link-op $n2 $n3 orient up
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n4 $n5 orient right
$ns duplex-link-op $n5 $n6 orient right-down
$ns duplex-link-op $n6 $n7 orient down
$ns duplex-link-op $n7 $n8 orient left-down
$ns duplex-link-op $n8 $n1 orient left
#Create a UDP agent and attach it to node n1
set udp0 [new Agent/UDP]
$ns attach-agent $n1 $udp0
#Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
#Create a Null agent (a traffic sink) and attach it to node n4
set null0 [new Agent/Null]
$ns attach-agent $n4 $null0
#Connect the traffic source with the traffic sink
```

$ns connect $udp0 $null0

#Schedule events for the CBR agent and the network dynamics $ns at 0.0 "$n1 label Source"

$ns at 0.0 "$n4 label Destination"

$ns at 0.5 "$cbr0 start"

$ns rtmodel-at 1.0 down $n3 $n4

$ns rtmodel-at 2.0 up $n3 $n4

$ns at 4.5 "$cbr0 stop"

#Call the finish procedure after 5 seconds of simulation time

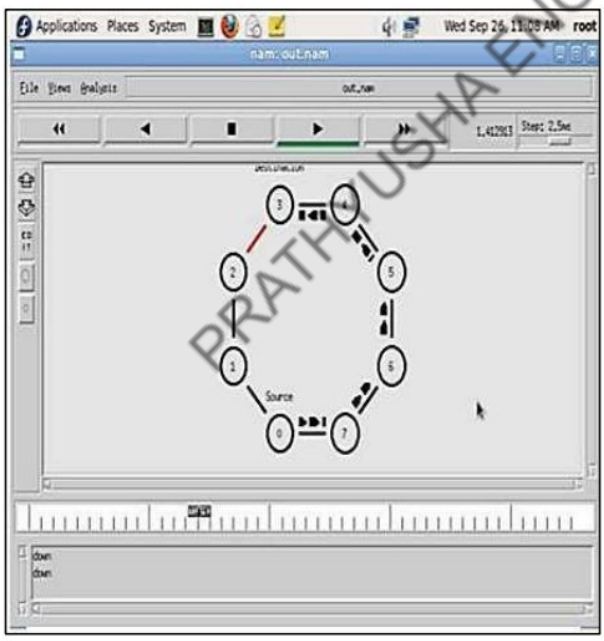
$ns at 5.0 "finish"

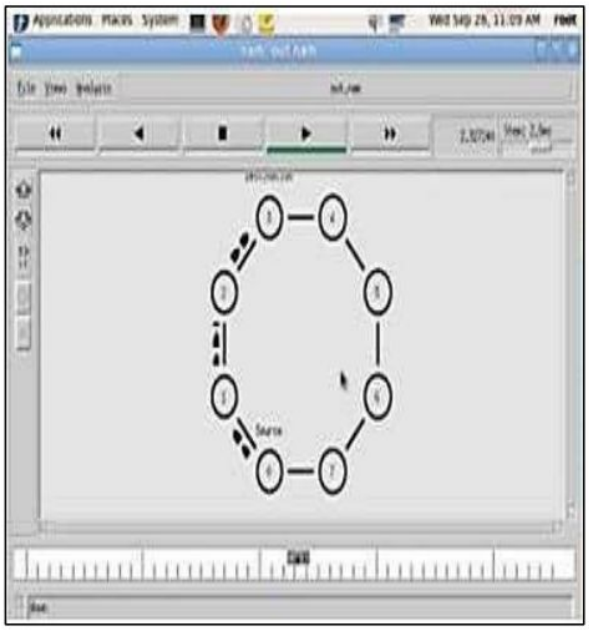
#Run the simulation

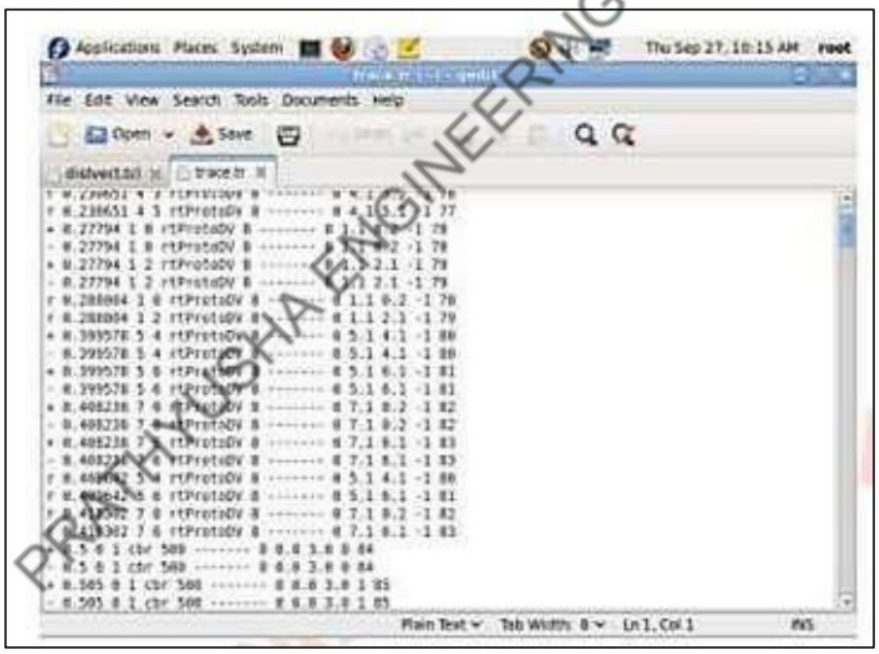
$ns run

OUTPUT

$ nsdistvect.tcl







RESULT:

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

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

Date:

AIM:

To implement error checking code using java.

ALGORITHM:

1. Start the Program
2. Given a bit string, append 0S to the end of it (the number of 0s is the same as the degree of the generator polynomial) let B(x) be the polynomial corresponding to B.
3. Divide B(x) by some agreed on polynomial G(x) (generator polynomial) and determine the remainder R(x). This division is to be done using Modulo 2 Division.
4. Define T(x) = B(x) –R(x)
5. (T(x)/G(x) => remainder 0)
6. Transmit T, the bit string corresponding to T(x).
7. Let T' represent the bit stream the receiver gets and T'(x) the associated polynomial. The receiver divides T1(x) by G(x). If there is a 0 remainder, the receiver concludes T = T' and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission
8. Stop the Program

PROGRAM

```
import java.io.*;
class crc_gen
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int[] data;
int[] div;
int[] divisor;
int[] rem;
int[] crc;
int data_bits, divisor_bits, tot_length;
System.out.println("Enter number of data bits : ");
data_bits=Integer.parseInt(br.readLine());
data=new int[data_bits];
System.out.println("Enter data bits : ");
 for(inti=0; i<data_bits; i++) data[i]=Integer.parseInt(br.readLine());
System.out.println("Enter number of bits in divisor : ");
divisor_bits=Integer.parseInt(br.readLine());
divisor=new int[divisor_bits];
System.out.println("Enter Divisor bits : ");

for(inti=0; i<divisor_bits; i++)
divisor[i]=Integer.parseInt(br.readLine());

 System.out.print("Data bits are : ");
for(inti=0; i<data_bits; i++)

System.out.print(data[i]);
System.out.println();
System.out.println("divisor bits are : ");
for(inti=0; i<divisor_bits; i++)
System.out.print(divisor[i]);
```

```
System.out.println();
 */
tot_length=data_bits+divisor_bits-1;
div=new int[tot_length];
rem=new int[tot_length];
crc=new int[tot_length];
/*------------ CRC GENERATION-------------------------*/
for(inti=0;i<data.length;i++)
 div[i]=data[i];
System.out.print("Dividend (after appending 0's) are : ");
for(inti=0; i<div.length; i++)
System.out.print(div[i]);
System.out.println();
for(int j=0; j<div.length; j++)
{
rem[j] = div[j];
}
rem=divide(div, divisor, rem);
 for(inti=0;i<div.length;i++)
{
//append dividend and remainder
crc[i]=(div[i]^rem[i]);
}
 System.out.println();
System.out.println("CRC code : ");
for(inti=0;i<crc.length;i++)
System.out.print(crc[i]);
```

```
/*------------ERROR DETECTION-------------*/

System.out.println();

System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(inti=0; i<crc.length; i++) crc[i]=Integer.parseInt(br.readLine());
System.out.print("crc bits are : ");

for(inti=0; i<crc.length; i++)
System.out.print(crc[i]);

System.out.println();
for(int j=0; j<crc.length; j++){

rem[j] = crc[j];
}

rem=divide(crc, divisor, rem);
for(inti=0; i<rem.length; i++)

{
if(rem[i]!=0)
System.out.println("Error");
break;
        }
        if(i==rem.length-1)

        System.out.println("No Error");
        }

        System.out.println("THANK YOU.....)");
        }

        staticint[] divide(int div[],int divisor[], int rem[])

        {
```

```
int cur=0;
while(true)
{
for(inti=0;i<divisor.length;i++)
rem[cur+i]=(rem[cur+i]^divisor[i]);
while(rem[cur]==0 && cur!=rem.length-1)
cur++;
if((rem.length-cur)<divisor.length)
break;
}
return rem;
}
}
```

OUTPUT

```
Enter number of data bits :
7
Enter data bits :
1
0
1
1
0
0
1
Enter number of bits in divisor :
3
Enter Divisor bits :
```

1

0

1

Dividend (after appending 0's) are : 101100100

CRC code :

101100111

Enter CRC code of 9 bits :

1

0

1

1

0

0

1

0

1

crc bits are : 101100101

Error

THANK YOU. ....)

BUILD SUCCESSFUL (total time: 1 minute 34 seconds)

RESULT:

Thus, the above program for error checking code using java was executed successfully.