KNOWLEDGE INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)

(Accredited by NAAC & NBA, Approved by AICTE, New Delhi and Affiliated To Anna University, Chennai)

KAKAPALAYAM (PO), SALEM – 637 504

Beyond Knowledge

RECORD NOTE BOOK

Register Number

Certified that this is the bonafide record of work done by Selvan/ Selvi……………………………………………………………of the……………………………. Semester ……………………………………………….…………………………… Branch during the year………………in the…………………………………………………………………………………….…… Laboratory.

Staff – In charge **Head of the Department**

Submitted for the University practical Examination on ……………………………….

Internal Examiner External Examiner

CONTENTS

Ex. No: 1	Networking Commands
Date:	

AIM

To learn about networking commands like tcpdump, netstat, ifconfig, nslookup and traceroute.

Networking Commands

1. **Netstat -** netstat (network statistics) is a command-line network utility that displays network connections for Transmission Control Protocol (both incoming and outgoing), routing tables, and a number of network interfaces (network interface controller or software-defined network interface) and network protocol statistics.

```
Command Prompt - netstat -a

Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Aseem>netstat -a

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            AseemVostro:0          LISTENING
  TCP    0.0.0.0:443            AseemVostro:0          LISTENING
  TCP    0.0.0.0:445            AseemVostro:0          LISTENING
  TCP    0.0.0.0:902            AseemVostro:0          LISTENING
  TCP    0.0.0.0:912            AseemVostro:0          LISTENING
  TCP    0.0.0.0:2869           AseemVostro:0          LISTENING
  TCP    0.0.0.0:5357           AseemVostro:0          LISTENING
```

- **Netstat -a** – provides a list of all available TCP and UDP connections
- **Netstat -e** – displays details of packets that have been sent
- **Netstat -n** – lists currently connected hosts
- **Netstat -p** – allow to specify what type of protocol you want to check
- **Netstat -r** – provides a list of routing tables
- **Netstat -s** – gives statistics on IPv4, IPv6, ICMP, TCP, etc.

2. **Ipconfig -** ipconfig (standing for "Internet Protocol configuration") is a console application program of some computer operating systems that displays all current TCP/IP network configuration values and refreshes Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS) settings.

```
Command Prompt

C:\Users\Aseem>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . : fios-router.home
   Link-local IPv6 Address . . . . . : fe80::257e:9cae:15eb:1922%10
   IPv4 Address. . . . . . . . . . . : 192.168.1.233
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
```

3. **Traceroute -** Traceroute is a command used in network troubleshooting for mapping the path packets travel through the network. The tool aids in the discovery of possible routes of information from source to destination. Additionally, the command also helps calculate the transfer times between points.

   When applied to network troubleshooting, traceroute helps locate where traffic slows down between the source and destination.

```
Command Prompt - tracert  helpdeskgeek.com

C:\Users\Aseem>tracert helpdeskgeek.com

Tracing route to helpdeskgeek.com [52.5.121.7]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  FIOS_Quantum_Gateway.fios-router.home [192.
  2     1 ms     2 ms     2 ms  lo0-100.BLTMMD-VFTTP-315.verizon-gni.net [1
  3     7 ms     7 ms     7 ms  B3315.BLTMMD-LCR-22.verizon-gni.net [130.81
  4     *        *        *     Request timed out.
  5     *        *        *     Request timed out.
  6     6 ms     6 ms     6 ms  0.et-11-0-2.GW13.IAD8.ALTER.NET [140.222.0.
  7     7 ms     5 ms     6 ms  neustar-gw.customer.alter.net [152.179.50.5
  8     *        *        *     Request timed out.
  9     *        *        *     Request timed out.
 10     5 ms    12 ms    19 ms  54.239.110.179
```

4. **NsLookup -** is a utility to query DNS tables, using this utility you can find what your DNS server is or any DNS server you specify, for example, nslookup google.com

Command Prompt

```
C:\Users\Aseem>nslookup google.com
Server:  FIOS_Quantum_Gateway.fios-router.home
Address:  192.168.1.1

Non-authoritative answer:
Name:    google.com
Addresses:  2607:f8b0:4004:805::200e
          172.217.7.142


C:\Users\Aseem>_
```

5. Tcpdump

tcpdump -i eth0 icmp

which will list ping traffic on interface eth0.

:~$ **sudo tcpdump -n icmp**

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode listening on enp7s0, link-type EN10MB (Ethernet), capture size 262144 bytes

11:34:21.590380 IP 10.10.1.217 > 10.10.1.30: ICMP echo request, id 27948, seq 1, length 64

11:34:21.590434 IP 10.10.1.30 > 10.10.1.217: ICMP echo reply, id 27948, seq 1, length 64

11:34:27.680307 IP 10.10.1.159 > 10.10.1.1: ICMP 10.10.1.189 udp port 59619 unreachable, length 115

Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus, the various networking commands have been learnt and executed successfully.

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

Aim:

To write a HTTP web client program to download a webpage using TCP sockets.

Algorithm:

Server Side

Step 1. Start.
Step 2. Import the necessary packages.
Step 3. Accept the request from the client.
Step 4. Establish socket connection between client and server.
Step 5. Read the image and send it to the client.
Step 6. Close the socket connection.
Step 7. Stop.

Client Side

Step 1. Start.
Step 2. Import the necessary packages.
Step 3. Establish the socket connection between client and server.
Step 4. Get the image from the server.
Step 5. Display the size of the image.
Step 6. Close the socket connection.
Step 7. Stop.

Program:

Server1.java

```
import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;

import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
```

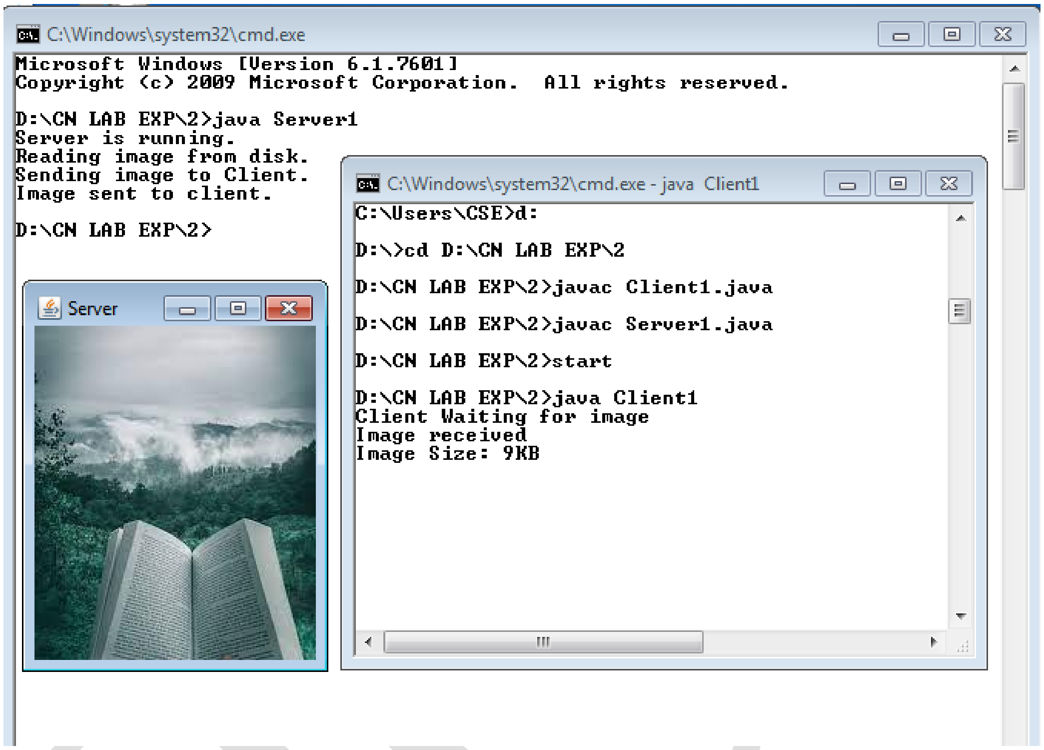
```
public class Server1
{
public static void main(String args[]) throws Exception
{
ServerSocket server=null;
Socket socket;
BufferedImage img = null;
server=new ServerSocket(4000);
System.out.println("Server is running. ");
socket=server.accept();
try
{
System.out.println("Reading image from disk. ");
img = ImageIO.read(new File("image1.jpg"));
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ImageIO.write(img, "jpg", baos);
baos.flush();
byte[] bytes = baos.toByteArray();
baos.close();
System.out.println("Sending image to Client. ");
OutputStream out = socket.getOutputStream();
DataOutputStream dos = new DataOutputStream(out);
dos.writeInt(bytes.length);
dos.write(bytes, 0, bytes.length);
System.out.println("Image sent to client. ");
dos.close();
out.close();
}
catch(Exception e)
{
System.out.println("Exception: " + e.getMessage());
socket.close();
}
socket.close();
}
}
```

Client1.java

```
import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;
```

```
public class Client1
{
public static void main(String args[]) throws Exception
{
Socket soc;
soc=new Socket("localhost",4000);
System.out.println("Client Waiting for image");
InputStream in = soc.getInputStream();
DataInputStream dis = new DataInputStream(in);
int len = dis.readInt();
System.out.println("Image received");
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



Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus, a HTTP web client program to download a webpage using TCP sockets has been written and executed successfully.

Ex. No: 3a	Applications using TCP sockets - Echo client and echo server
Date:	

Aim:

To write a Java program to implement Echo Client and Echo Server using TCP sockets.

Algorithm:

Server Side

Step 1. Start.

Step 2. Import the necessary packages.

Step 3. Under EchoServer class, create a server socket to communicate with the client using ServerSocket() constructor.

Step 4. Accept the request from the client.

Step 5. Establish socket connection between client and server using BufferedReader.

Step 6. Echo the messages back to the client.

Step 7. Close the socket connection.

Step 8. Stop.

Client Side

Step 1. Start.

Step 2. Import the necessary packages.

Step 3. Under EchoClient class, create a new socket with IP address of the server system using Socket() constructor.

Step 4. Establish the socket connection between client and server using BufferedReader.

Step 5. Send messages to the server using getOutputStream() method.

Step 6. Display the message that is echoed back from the server.

Step 7. Close the socket connection.

Step 8. Stop.

Program:

tcpechoserver .java

```
import java.net.*;
import java.io.*;
public class tcpechoserver
{
public static void main(String[] arg) throws IOException
{
ServerSocket sock = null;
BufferedReader fromClient = null;
OutputStreamWriter toClient = null;
Socket client = null;
try
{
sock = new ServerSocket(4000);
System.out.println("Server Ready");
client = sock.accept();
System.out.println("Client Connected");
fromClient = new BufferedReader(new
InputStreamReader(client.getInputStream()));
toClient = new
OutputStreamWriter(client.getOutputStream());
String line;
while (true)
{
line = fromClient.readLine();
if ( (line == null) || line.equals("bye"))
break;
System.out.println ("Client [ " + line + "]");
toClient.write("Server [ "+ line +" ]\n");
toClient.flush();
}
fromClient.close();
toClient.close();
client.close();
sock.close();
System.out.println("Client Disconnected");
}
catch (IOException ioe)
{
```

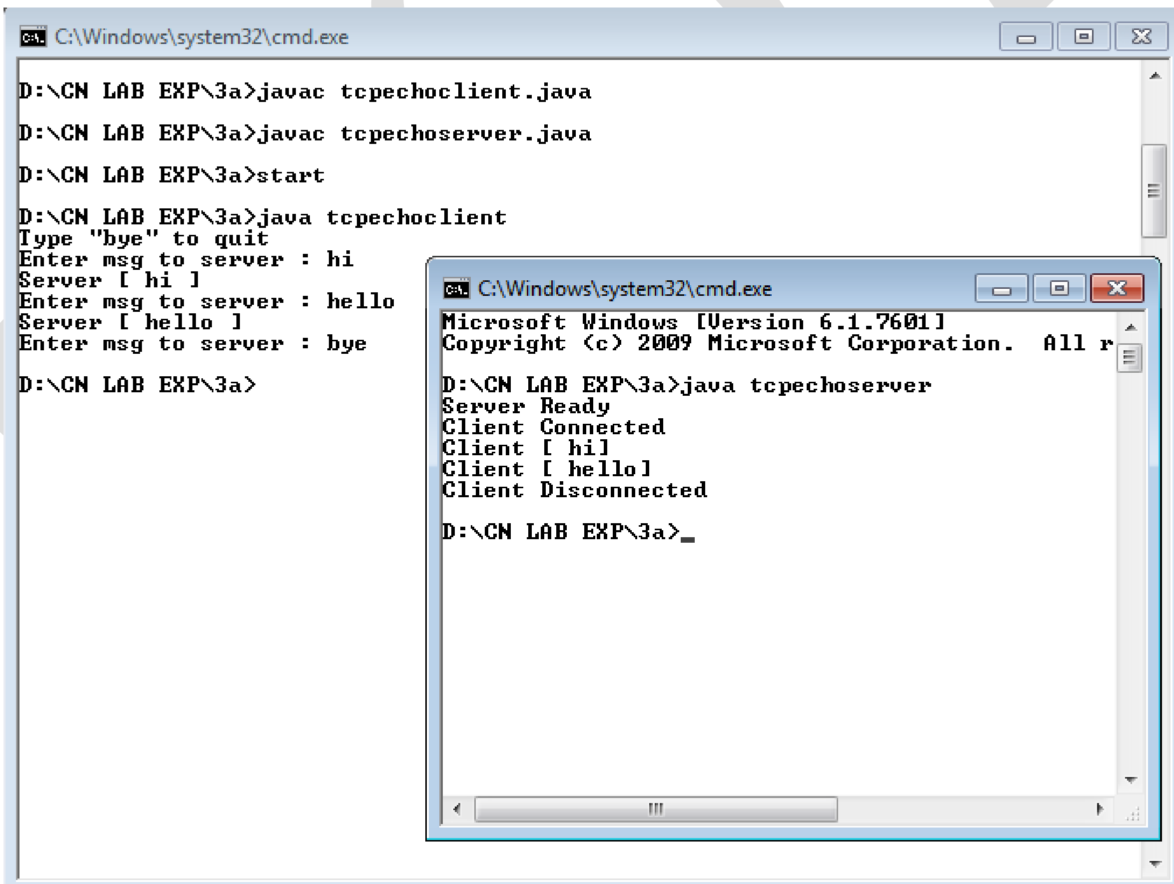
```
System.err.println(ioe);
}
}
}
```

tcpechoclient.java

```
import java.net.*;
import java.io.*;
public class tcpechoclient
{
public static void main(String[] args) throws IOException
{
BufferedReader fromServer = null, fromUser = null;
PrintWriter toServer = null;
Socket sock = null;
try
{
if (args.length == 0)
 sock = new Socket(InetAddress.getLocalHost(),4000);
else
 sock = new  Socket(InetAddress.getByName(args[0]),4000);
fromServer = new BufferedReader(newInputStreamReader(sock.getInputStream()));
fromUser = new BufferedReader(newInputStreamReader(System.in));
toServer = new PrintWriter(sock.getOutputStream(),true);
String Usrmsg, Srvmsg;
System.out.println("Type \"bye\" to quit");
while (true)
{
System.out.print("Enter msg to server : ");
Usrmsg = fromUser.readLine();
if (Usrmsg==null || Usrmsg.equals("bye"))
{
toServer.println("bye");
break;
}
else
toServer.println(Usrmsg);
Srvmsg = fromServer.readLine();
System.out.println(Srvmsg);
}
fromUser.close();
```

```
fromServer.close();
toServer.close();
sock.close();
}
catch (IOException ioe)
{
System.err.println(ioe);
}
}
}
```

Output:



Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus, a Java program to implement Echo Client and Echo Server using TCP sockets has been written and executed successfully.

Ex. No:3b	Applications using TCP sockets – Chat Server with client
Date:	

Aim:

To write a Java program to implement the Chat application between Client and Server using Sockets.

Algorithm:

Server Side

Step 1. Start.
Step 2. Create a server socket.
Step 3. Wait for client to be connected.
Step 4. Read Client's message and display it.
Step 5. Get a message from user and send it to client.
Step 6. Repeat steps 3-4 until the client sends "end".
Step 7. Close all streams.
Step 8. Close the server and client socket.
Step 9. Stop.

Client Side

Step 1. Start.
Step 2. Create a client socket and establish connection with the server.
Step 3. Get a message from user and send it to server.
Step 4. Read server's response and display it.
Step 5. Repeat steps 2-3 until chat is terminated with "end" message.
Step 6. Close all input/output streams.
Step 7. Close the client socket.
Step 8. Stop.

Program:

<u>chatServer.java</u>

```
import java.net.*;
import java.io.*;
class chatServer
{
public static void main (String args[]) throws Exception
{
ServerSocket ss  ;
DataInputStream uin ;
DataInputStream din ;
```

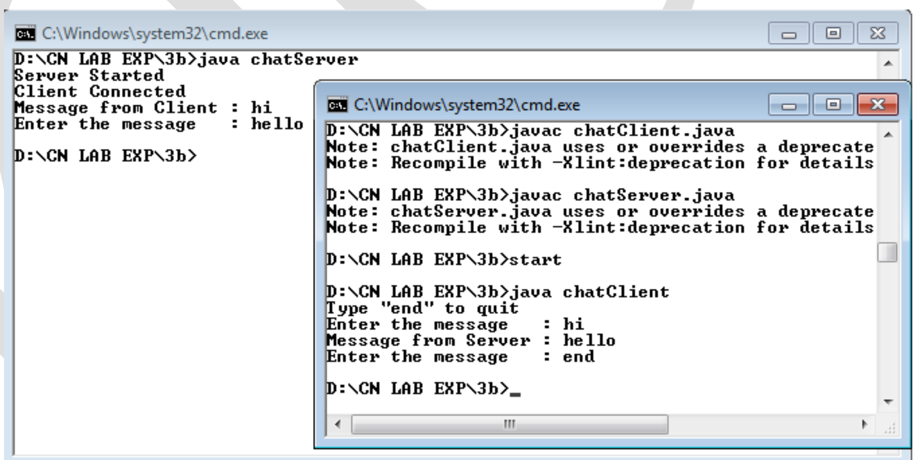
```
PrintStream p   ;
Socket s   ;
try
{
ss = new ServerSocket (1711) ;
System.out.println("Server Started");
s  = ss.accept();
System.out.println("Client Connected");
din = new DataInputStream (s.getInputStream() ) ;
uin = new DataInputStream (System.in) ;
p   = new PrintStream (s.getOutputStream() ) ;
String str ;
String str1 ;
while(true)
{
str = din.readLine() ;
while ( !(str.equalsIgnoreCase("end") ) )
{
System.out.println ("Message from Client : " + str ) ;
System.out.print ("Enter the message   : " ) ;
str1 = uin.readLine();
p.println(str1);
str = din.readLine();
}
}
}
catch(IOException e)
{
}
}
}
```

chatClient.java

```
import java.net.*;
import java.io.*;
class chatClient
{
public static void main (String args[]) throws Exception
{
Socket  s ;
DataInputStream dis;
PrintStream p;
```

```java
InetAddress i = InetAddress.getLocalHost();
String msg ;
s   = new Socket (i,1711) ;
dis = new DataInputStream(s.getInputStream());
p   = new PrintStream(s.getOutputStream());
DataInputStream uin = new DataInputStream (System.in) ;
String str;
System.out.println("Type \"end\" to quit");
System.out.print( "Enter the message   : " ) ;
str = uin.readLine();
while(!(str.equalsIgnoreCase("end")))
 {
p.println(str);
msg=dis.readLine();
System.out.println( "Message from Server : " + msg ) ;
System.out.print( "Enter the message   : " ) ;
str = uin.readLine() ;
}
}
}
```

Output:



Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus, a Java program to implement the Chat application between Client and Server using Sockets has been written and executed successfully.

Ex. No:4	Simulation of DNS using UDP sockets.
Date:	

Aim:

To write a Java program for simulation of DNS using UDP sockets.

Algorithm:

Server Side

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

Client

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

Program:

Serverdns12.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverdns12
{
public static void main(String args[])
{
try
{
System.out.println("Server Ready");
System.out.println("Press Ctrl + C to Quit");
DatagramSocket server=new DatagramSocket(1309);
while(true)
{
```

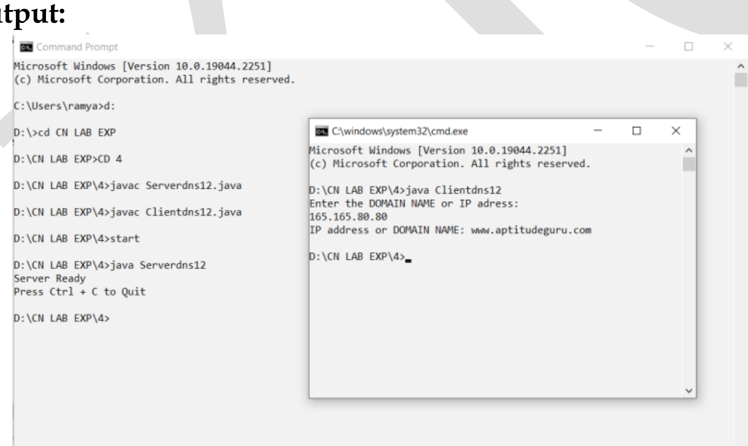
```
byte[] sendbyte=new byte[1024];
byte[] receivebyte=new byte[1024];
DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);
server.receive(receiver);
String str=new String(receiver.getData());
String s=str.trim();
InetAddress addr=receiver.getAddress();
int port=receiver.getPort();
String ip[]={"165.165.80.80","165.165.79.1"};
String name[]={"www.aptitudeguru.com","www.downloadcyclone.blogspot.com"};
for(int i=0;i<ip.length;i++)
{
if(s.equals(ip[i]))
{
sendbyte=name[i].getBytes();
DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr,port);
server.send(sender);
break;
}
else if(s.equals(name[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);
}
}
}
```

Client dns12.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientdns12
{
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 DOMAIN NAME or IP adress:");
String str=in.readLine();
sendbyte=str.getBytes();
DatagramPacket sender=new DatagramPacket(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("IP address or DOMAIN NAME: "+s.trim());
client.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

Output:



Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus, a Java program simulation of DNS using UDP sockets has been written and executed successfully.

Ex. NO: 5	Use a tool like Wireshark to Capture Packets and examine the packets
Date:	

Aim:

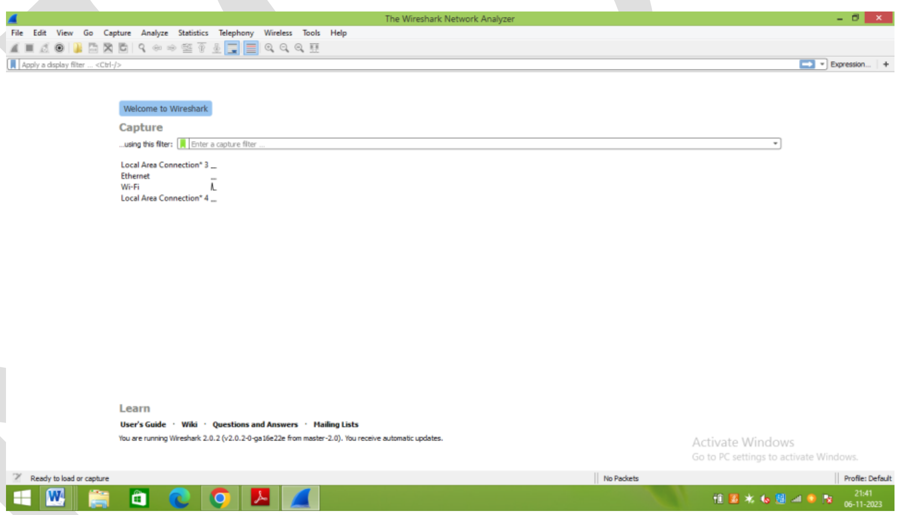
To capture and examine the packets using wireshark tool.

Theory

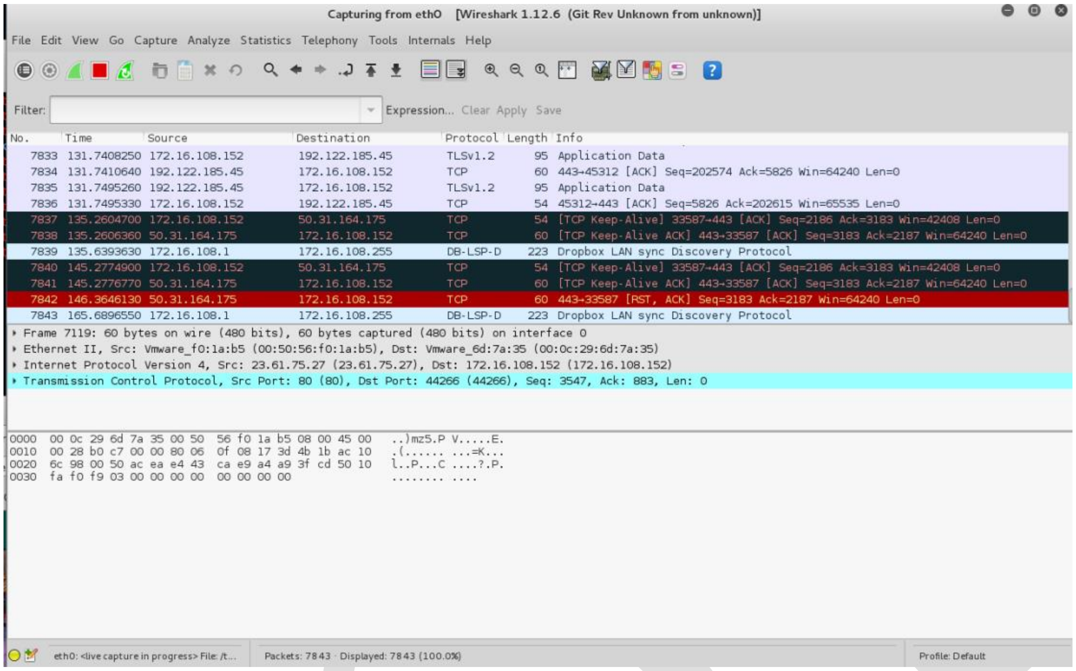
Wireshark

Wireshark is a free open- source network protocol analyzer. It is used for network troubleshooting and communication protocol analysis. Wireshark captures network packets in real time and display them in human-readable format. It provides many advanced features including live capture and offline analysis, three-pane packet browser, coloring rules for analysis.

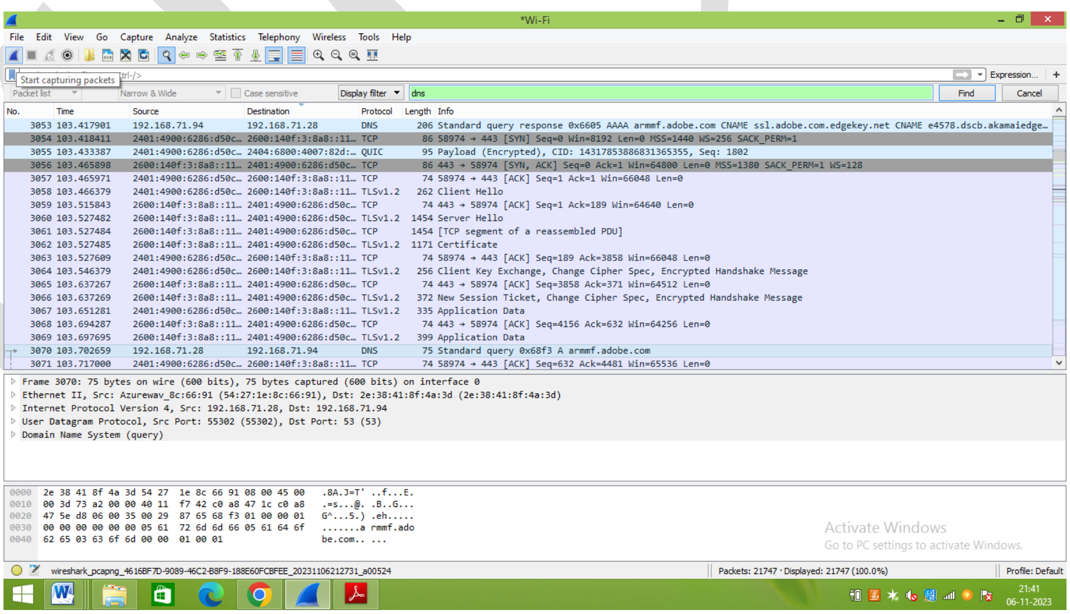
- Start up the Wireshark program (select an interface and press start to capture packets).



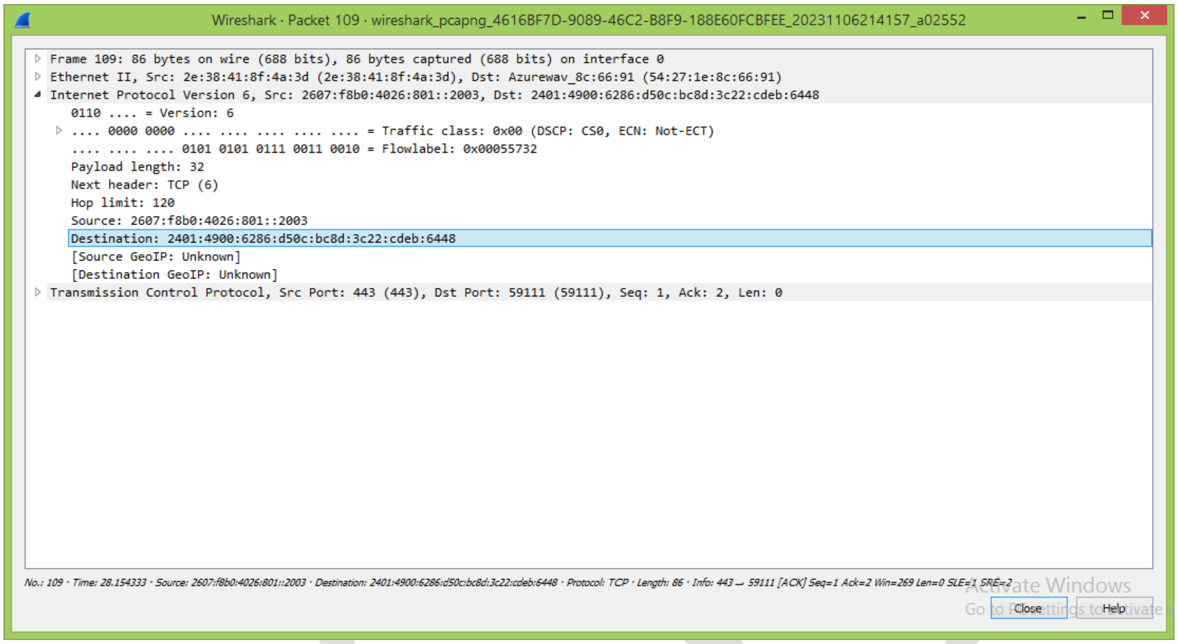
- Start up your favorite browser
- After your browser has displayed the page, stop Wireshark packet capture by selecting stop in the Wireshark capture window. This will cause the Wireshark capture window to disappear and the main Wireshark window to display all packets captured



Here tcp packets and frame formats are displayed. In this window sequence number, acknowledgement number , header length, window size, sequence Ack no.

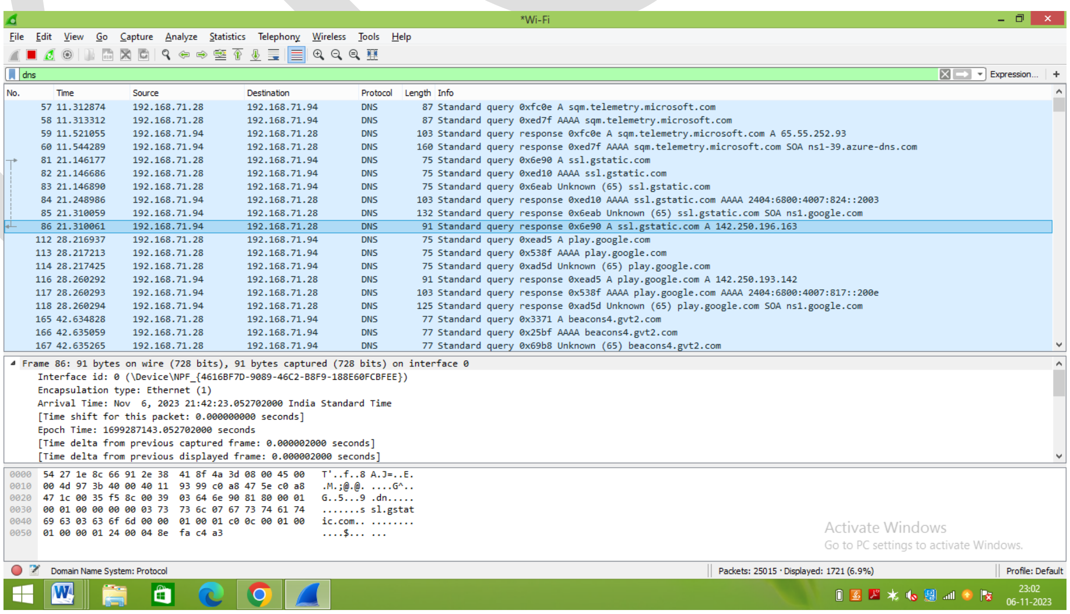


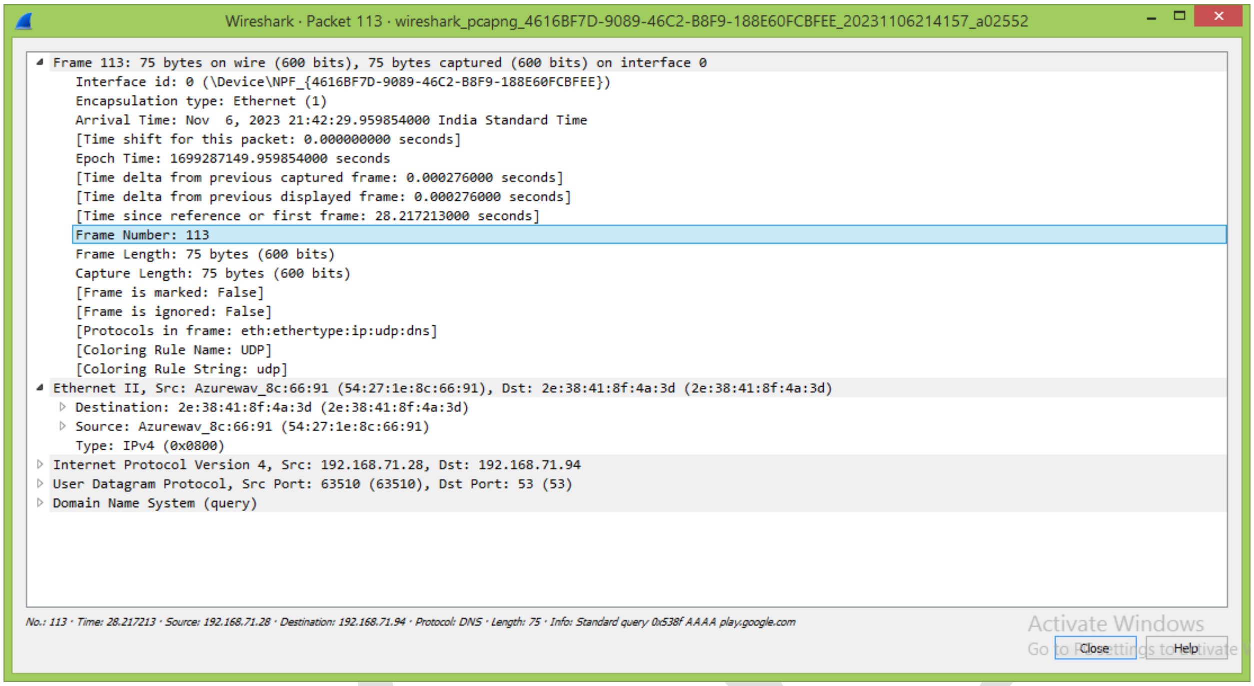
Here is the format of ipV6 protocol in tcp



Domain Name System

Domain Name System is the system used to resolve store information about domain names including IP addresses, mail servers, and other information.





This picture shows the details of frame of DNS, Ethernet, IP, User Datagram Protocol, Dns (query) 113 frames is used, ipv4 addressing is used and have format of each protocol .

Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result

Thus captured and examined the packets using wireshark tool.

Ex. No: 6a	Write a code simulating ARP protocol.
Date:	

Aim:

To write a Java program for simulating ARP protocol using TCP.

Algorithm:

Server Side

Step 1. Start.
Step 2. Create a server socket and bind it to port.
Step 3. Listen for new connection and when a connection arrives, accept it.
Step 4. Read the logical address sent by the client.
Step 5. Send the corresponding physical address to the client.
Step 6. Close the server socket.
Step 7. Stop.

Client Side

Step 1. Start.
Step 2. Create a client socket and connect it to the server.
Step 3. Send the logical address to the server.
Step 4. Display the corresponding address sent by the server.
Step 5. Close the input and output streams.
Step 6. Close the client socket.
Step 7. Stop.

Program:

Serverarp.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverarp
{
public static void main(String args[])
{
try
{
System.out.println("Server Ready");
ServerSocket obj=new ServerSocket(5000);
Socket obj1=obj.accept();
while(true)
{
DataInputStream din=new DataInputStream(obj1.getInputStream());
```

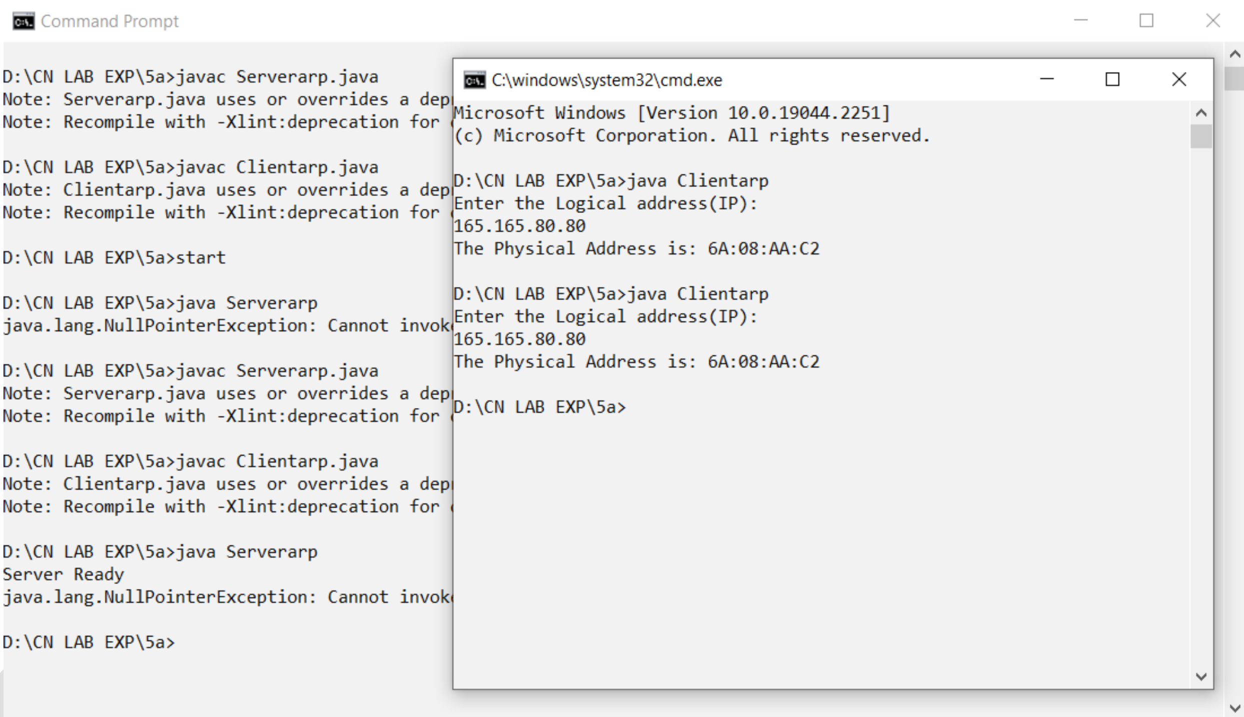
```
DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());
String str=din.readLine();
String ip[]={"165.165.80.80","165.165.79.1"};
String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
for(int i=0;i<ip.length;i++)
{
if(str.equals(ip[i]))
{
dout.writeBytes(mac[i]+'\n');
break;
}
}
obj.close();
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

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",5000);
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);
}
}
}
```

Output:



Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus, a Java program to simulate ARP protocols using TCP has been written and executed successfully.

Ex. No: 6b	**Write a code simulating RARP protocol.**
Date:	

Aim:

To write a Java program for simulating RARP protocol using UDP.

Algorithm:

Server Side

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

Client Side

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

Program:

<u>Serverrarp12.java</u>

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverrarp12
{
public static void main(String args[])
{
try
{
System.out.println("Server Ready");
DatagramSocket server=new DatagramSocket(1309);
while(true)
{
byte[] sendbyte=new byte[1024];
byte[] receivebyte=new byte[1024];
DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);
```

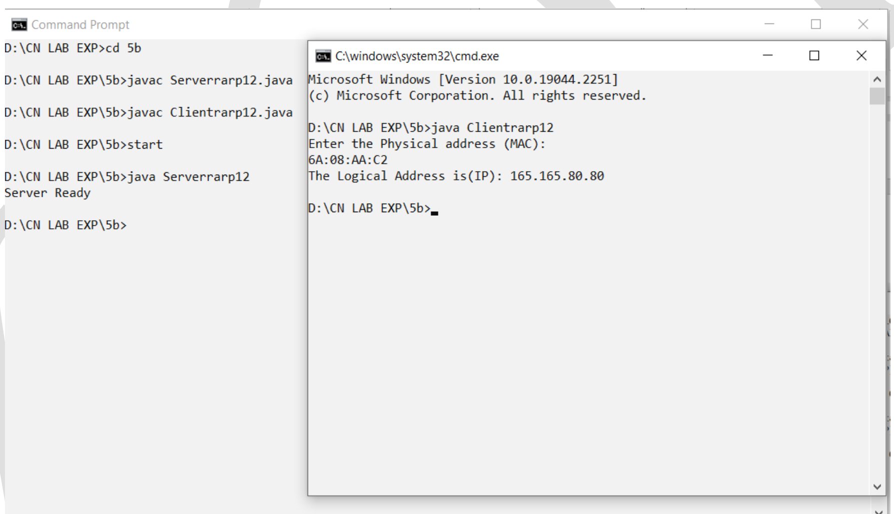
```
server.receive(receiver);
String str=new String(receiver.getData());
String s=str.trim();
InetAddress addr=receiver.getAddress();
int port=receiver.getPort();
String ip[]={"165.165.80.80","165.165.79.1"};
String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
for(int i=0;i<ip.length;i++)
{
if(s.equals(mac[i]))
{
sendbyte=ip[i].getBytes();
DatagramPacket sender = new
DatagramPacket(sendbyte,sendbyte.length,addr,port);
server.send(sender);
break;
}
}
break;
}
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

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();
```

```java
DatagramPacket sender=new DatagramPacket(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);
}
}
}
```

Output:



Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus, a Java program to simulate RARP protocols using UDP has been written and executed successfully.

Ex. No: 7a	Study of Network Simulator(NS2)
Date:	

Aim:

To study about the Network Simulator 2(NS2).

Theory:

Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989. Ever since, several revolutions and revisions have marked the growing maturity of the tool, thanks to substantial contributions from the players in the field. Among these are the University of California and Cornell University who developed the REAL network simulator,1 the foundation which NS is based on. Since
1995 the Defense Advanced Research Projects Agency (DARPA) supported development of NS through the Virtual Inter Network Testbed (VINT) project . Currently the National Science Foundation (NSF) has joined the ride in development. Last but not the least, the group of Researchers
and developers in the community are constantly working to keep NS2 strong and versatile.

Basic Architecture:

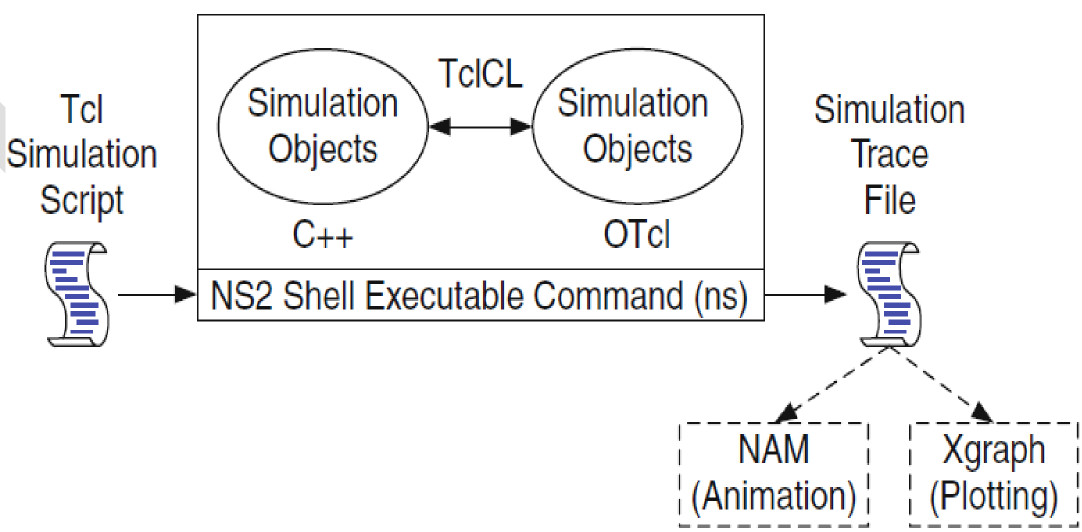


Figure:1 Basic Architecture of NS2

Figure 2.1 shows the basic architecture of NS2. NS2 provides users with executable command ns which take on input argument, the name of a Tcl simulation scripting file. Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS2 executable command ns.

In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the

C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).

The C++ and the OTcl are linked together using TclCL. Mapped to a C++ object, variables in the OTcl domains are sometimes referred to as handles. Conceptually, a handle (e.g., n as a Node handle) is just a string (e.g.,_o10) in the OTcl domain, and does not contain any functionality. Instead, the functionality (e.g., receiving a packet) is defined in the mapped C++ object (e.g., of class Connector). In the OTcl domain, a handle acts as a frontend which interacts with users and other OTcl objects. It may define its own procedures and variables to facilitate the interaction. Note that the member procedures and variables in the OTcl domain are called instance procedures (instprocs) and instance variables (instvars), respectively. Before proceeding further, the readers are encouraged to learn C++ and OTcl languages.

NS2 provides a large number of built-in C++ objects. It is advisable to use these C++ objects to set up a simulation using a Tcl simulation script. However, advance users may find these objects insufficient. They need to develop their own C++ objects, and use a OTcl configuration interface to put together these objects. After simulation, NS2 outputs either text-based or animation-based simulation results. To interpret these results graphically and interactively, tools such as NAM (Network AniMator) and XGraph are used. To analyze a particular behaviour of the network, users can extract a relevant subset of text-based data and transform it to a more conceivable presentation.

Concept Overview:

NS uses two languages because simulator has two different kinds of things it needs to do. On one hand, detailed simulations of protocols requires a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios. In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. ns meets both of these needs with two languages, C++ and OTcl.

Tcl scripting

Tcl is a general purpose scripting language. [Interpreter]

- Tcl runs on most of the platforms such as Unix, Windows, and Mac.
- The strength of Tcl is its simplicity.
- It is not necessary to declare a data type for variable prior to the usage.

Basics of TCL

Syntax: command arg1 arg2 arg3

Hello World!

puts stdout{Hello, World!} Hello, World!

Variables Command Substitution

set a 5 set len [string length foobar]

set b $a set len [expr [string length foobar] + 9]

Wired TCL Script Components

- Create the event scheduler
- Open new files & turn on the tracing
- Create the nodes
- Setup the links
- Configure the traffic type (e.g., TCP, UDP, etc)

- Set the time of traffic generation (e.g., CBR, FTP)
- Terminate the simulation

NS Simulator Preliminaries

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

set ns [new Simulator]

which is thus the first line in the tcl script. This line declares a new variable as using the set command. In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.

In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), need to create the files using ―open command:

#Open the Trace file

set tracefile1 [open out.tr w]
$ns trace-all $tracefile1

#Open the NAM trace file

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

The above creates a dta trace file called out.tr and a nam visualization trace file called out.nam. Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called ―tracefile1 and ―namfile respectively. Remark that they begins with a # symbol. The second line open the file ―out.tr to be used for writing, declared with the letter ―w. The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

Define a "finish" procedure

Proc finish { } {
global ns tracefile1 namfile
$ns flush-trace
Close $tracefile1
Close $namfile
Exec nam out.nam &
Exit 0
}

Definition of a network of links and nodes

The way to define a node is

set n0 [$ns node]

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

$ns duplex-link $n0 $n2 10Mb 10ms DropTail

which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.

To define a directional link instead of a bi-directional one, replace ―duplex-link by ―simplex-link.

In ns, an output queue of a node is implemented as a part of each link whose input is that node. Also define the buffer capacity of the queue related to each link.

Example

#set Queue Size of link (n0-n2) to 20
$ns queue-limit $n0 $n2 20

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received. There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

set tcp [new Agent/TCP]

The command $ns attach-agent $n0 $tcp defines the source node of the tcp connection. The command set sink [new Agent /TCPSink] defines the behavior of the destination node of TCP and assigns to it a pointer called sink.

#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

The below shows the definition of a CBR application using a UDP agent.

The command **$ns attach-agent $n4 $sink** defines the destination node. The command $ns connect $tcp $sink finally makes the TCP connection between the source and destination nodes.

set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetsize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command **$tcp set packetSize_ 552**. When there are several flows, distinguish them with different colors in the visualization part. This is done by the command **$tcp set fid_ 1** that assigns to the TCP connection a flow identification of -1. Assign the flow identification of ―2‖ to the UDP connection.

Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus the Network Simulator2 (NS2) has been studied successfully.

Ex. No: 7b	Simulation of Congestion Control Algorithms using NS
Date:	

Aim:

To simulate the TCP congestion control mechanism in NS2.

Algorithm:

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

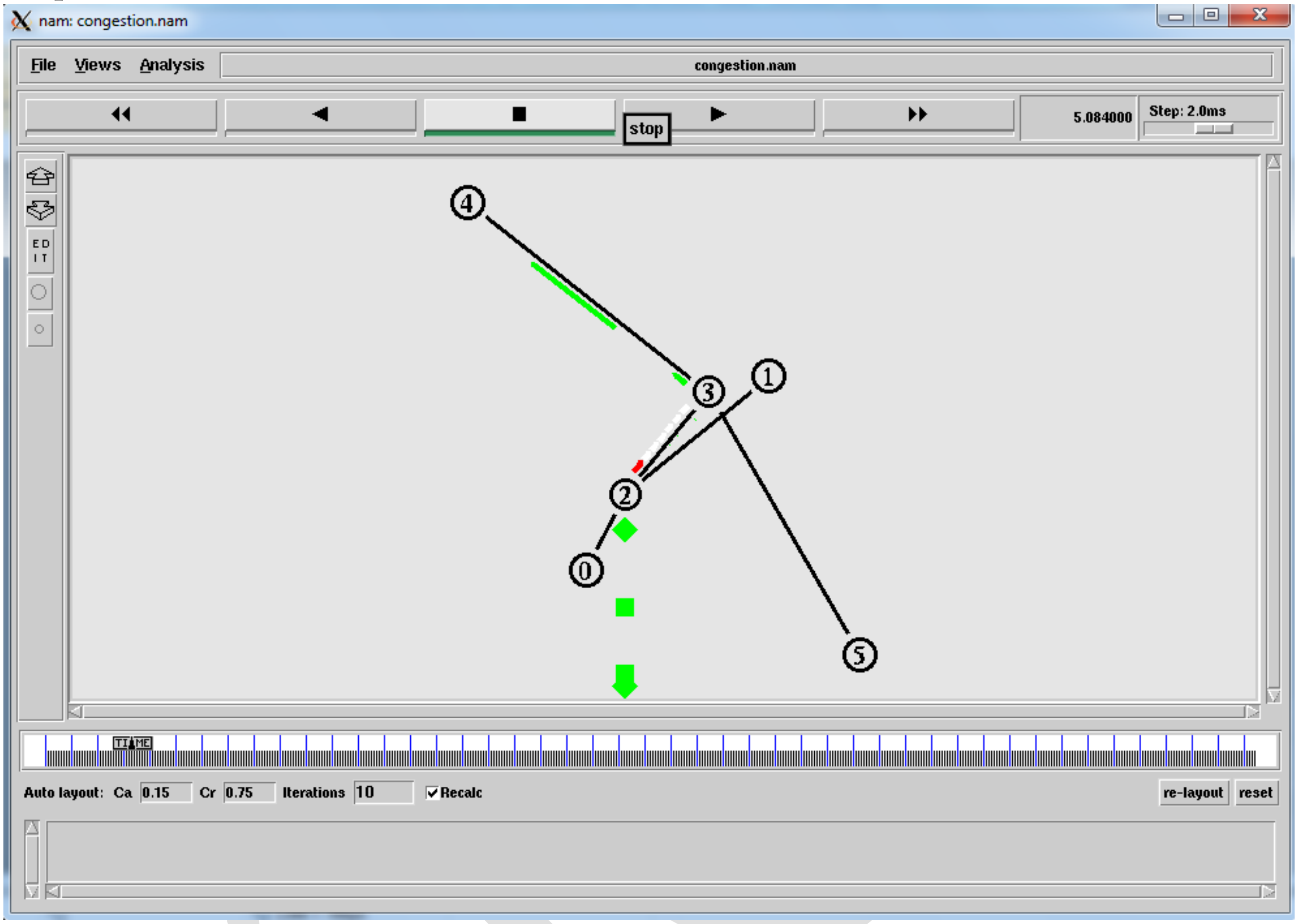
Program:

```
set ns [new Simulator]
set f [ open congestion.tr w ]
$ns trace-all $f
set nf [ open congestion.nam w ]
$ns namtrace-all $nf
$ns color 1 Red
$ns color 2 Blue
$ns color 3 White
$ns color 4 Green
#to create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
# to create the link between the nodes with bandwidth, delay and queue
$ns duplex-link $n0 $n2   2Mb  10ms DropTail
$ns duplex-link $n1 $n2   2Mb  10ms DropTail
$ns duplex-link $n2 $n3 0.3Mb 200ms DropTail
$ns duplex-link $n3 $n4 0.5Mb  40ms DropTail
$ns duplex-link $n3 $n5 0.5Mb  30ms DropTail
```

```
# Sending node with agent as Reno Agent
set tcp1 [new Agent/TCP/Reno]
$ns attach-agent $n0 $tcp1
set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp2
set tcp3 [new Agent/TCP/Reno]
$ns attach-agent $n2 $tcp3
set tcp4 [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp4
$tcp1 set fid_ 1
$tcp2 set fid_ 2
$tcp3 set fid_ 3
$tcp4 set fid_ 4
# receiving (sink) node
set sink1 [new Agent/TCPSink]
$ns attach-agent $n4 $sink1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n5 $sink2
set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
set sink4 [new Agent/TCPSink]
$ns attach-agent $n4 $sink4
# establish the traffic between the source and sink
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2
$ns connect $tcp3 $sink3
$ns connect $tcp4 $sink4
# Setup a FTP traffic generator on "tcp"
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP
set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
$ftp3 set type_ FTP
set ftp4 [new Application/FTP]
$ftp4 attach-agent $tcp4
$ftp4 set type_ FTP
# RTT Calculation Using Ping ------------------------------------------------
set p0 [new Agent/Ping]
$ns attach-agent $n0 $p0
set p1 [new Agent/Ping]
```

```
$ns attach-agent $n4 $p1
#Connect the two agents
$ns connect $p0 $p1
# Method call from ping.cc file
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] received ping answer from \
$from with round-trip-time $rtt ms."
}
# start/stop the traffic
$ns at 0.2 "$p0 send"
$ns at 0.3 "$p1 send"
$ns at 0.5   "$ftp1 start"
$ns at 0.6   "$ftp2 start"
$ns at 0.7 "$ftp3 start"
$ns at 0.8   "$ftp4 start"
$ns at 66.0 "$ftp4 stop"
$ns at 67.0  "$ftp3 stop"
$ns at 68.0 "$ftp2 stop"
$ns at 70.0 "$ftp1 stop"
$ns at 70.1 "$p0 send"
$ns at 70.2 "$p1 send"
# Set simulation end time
$ns at 80.0 "finish"
# procedure to plot the congestion window
# cwnd_ used from tcp-reno.cc file
proc plotWindow {tcpSource outfile} {
global ns
set now [$ns now]
set cwnd_ [$tcpSource set cwnd_]
# the data is recorded in a file called congestion.xg.
puts  $outfile  "$now $cwnd_"
$ns at [expr $now+0.1] "plotWindow $tcpSource  $outfile"
}
set outfile [open  "congestion.xg"  w]
$ns  at  0.0  "plotWindow $tcp1  $outfile"
proc finish {} {
 exec nam congestion.nam &
 exec xgraph congestion.xg -geometry 300x300 &
exit 0
}
# Run simulation
$ns run
```

Output:



Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus the tcl script to simulate TCP congestion control algorithms has been written and executed successfully.

Ex. No: 8	Study of TCP/UDP performance using simulation tool
Date:	

Aim:

To study the performance of TCP/UDP using simulation tool.

Algorithm:

TCP:

Step 10. Create a Simulator object.
Step 11. Set routing as dynamic.
Step 12. Open the trace and nam trace files.
Step 13. Define the finish procedure.
Step 14. Create nodes and the links between them.
Step 15. Create the agents and attach them to the nodes.
Step 16. Create the applications and attach them to the tcp agent.
Step 17. Connect tcp and tcp sink.
Step 18. Run the simulation.

UDP:

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

Program:

<u>TCP program</u>

```
set ns [new Simulator]
set f [open tcpout.tr w]
$ns trace-all $f
set nf [open tcpout.nam w]
$ns namtrace-all $nf
$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]
$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 tcp [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"
proc finish {} {
global ns f nf
$ns flush-trace
close $f
close $nf
puts "Running nam.."
#exec xgraph tcpout.tr -geometry 600x800 &
exec nam tcpout.nam &
exit 0
}
$ns run
```

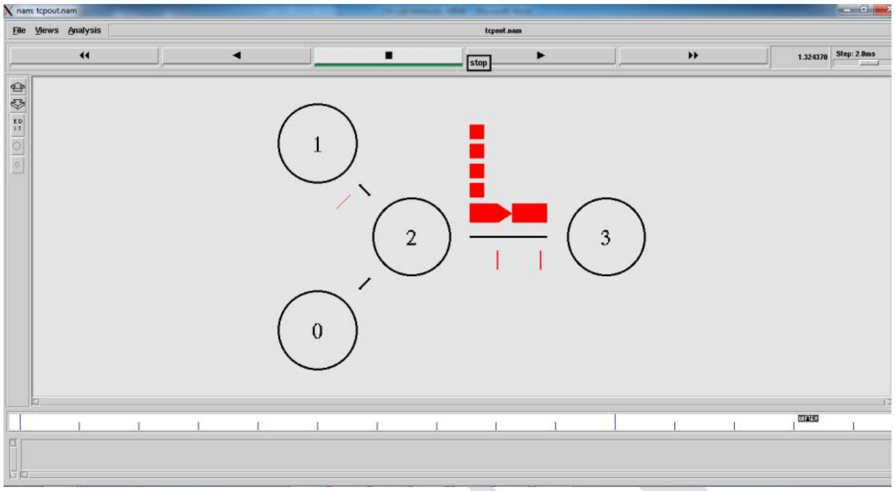
UDP Program

```
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
set nf [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.."
exec nam udpout.nam &
exit 0
}
$ns run
```

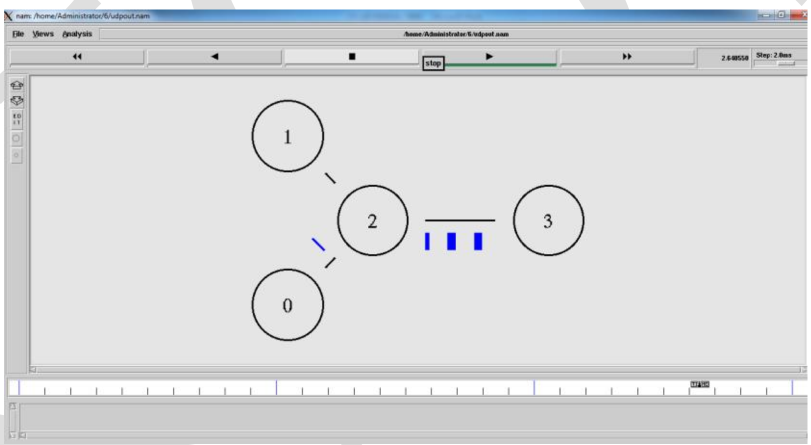
Output:

TCP Output



UDP Output

```
Administrator@CSE-159 ~/6
$ ns udpperform.tcl
210
0.0037499999999999999
Running nam..
```



Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus the performance of TCP/UDP using simulation tool has been done and executed successfully.

Ex. No:9a	Simulation of Distance Vector Routing Algorithm
Date:	

Aim:

To simulate the Distance Vector Routing Algorithm using simulation tool.

Algorithm:

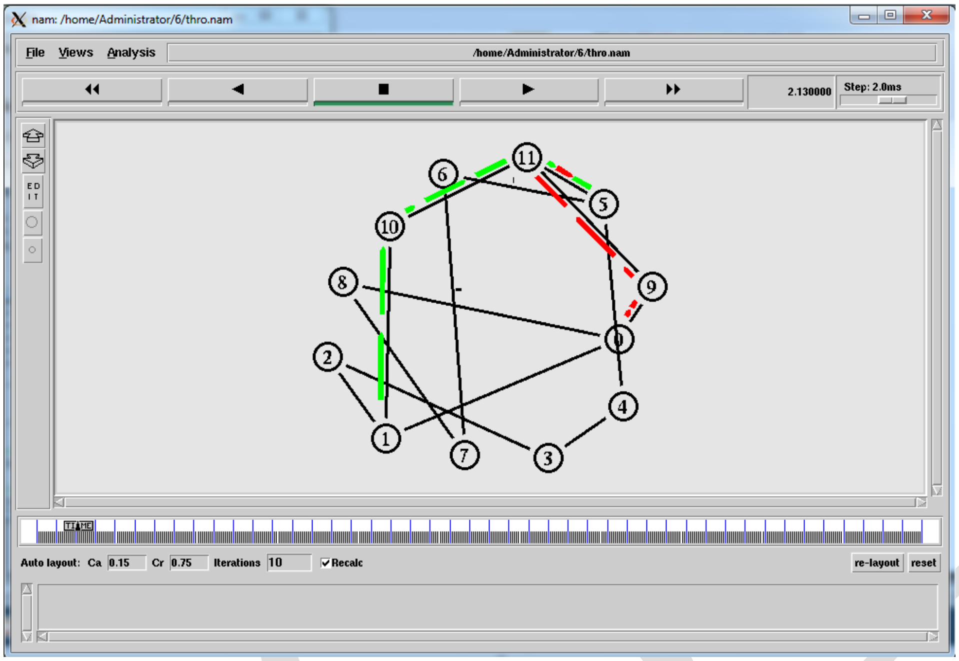
Step 1. Create a simulator object.
Step 2. Set routing protocol to Distance Vector routing.
Step 3. Trace packets on all links onto NAM trace and text trace file.
Step 4. Define finish procedure to close files, flush tracing and run NAM.
Step 5. Create the nodes.
Step 6. Specify the link characteristics between nodes.
Step 7. Add UDP agent for node n1.
Step 8. Create CBR traffic on top of UDP and set traffic parameters.
Step 9. Connect source and the sink.
Step 10. Schedule the events.
Step 11. Observe the traffic route when link is up and down.
Step 12. Stop.

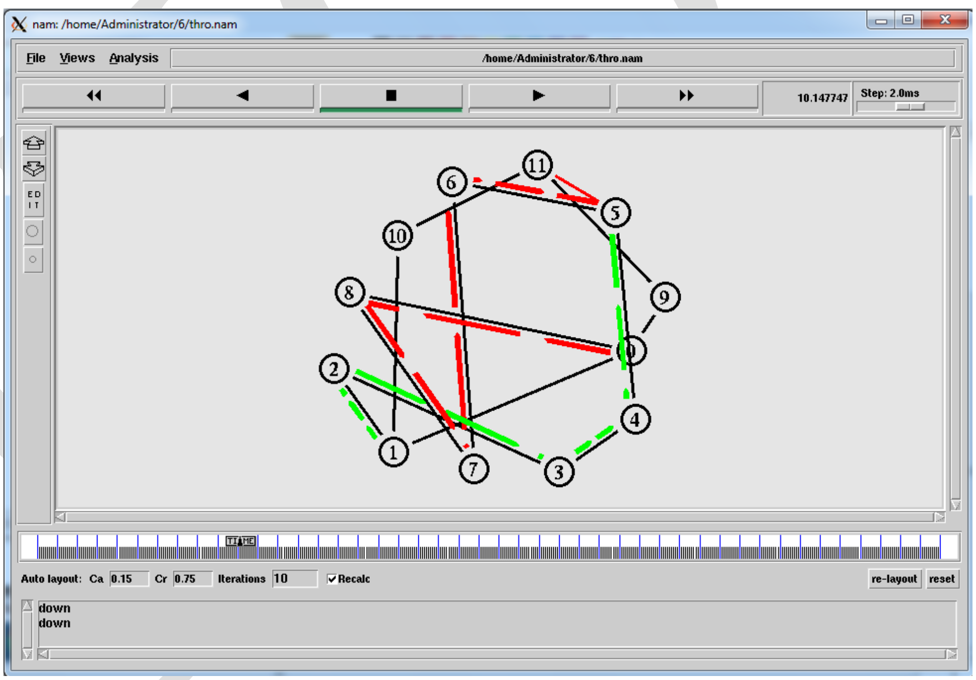
Program:

```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]
$ns namtrace-all $nf
proc finish { } {
global ns nr nf
 $ns flush-trace
close $nf
close $nr
exec nam thro.nam &
exit 0
 }
for { set i 0 } { $i < 12} { incr i 1 } {
set n($i) [$ns node]}
for {set i 0} {$i < 8} {incr i} {
$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail }
$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
```

```
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0
set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0
$ns rtproto DV
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)
$udp0 set fid_ 1
$udp1 set fid_ 2
$ns color 1 Red
$ns color 2 Green
$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"
$ns at 45 "finish"
$ns run
```

Output:





Result:

Thus the tcl script to simulate Distance Vector Routing algorithm has been written and executed successfully.

Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Ex. No: 9b	**Simulation of Link State Routing Algorithm**
Date:	

Aim:

To simulate the Link State Routing Algorithm using simulation tool.

Algorithm:

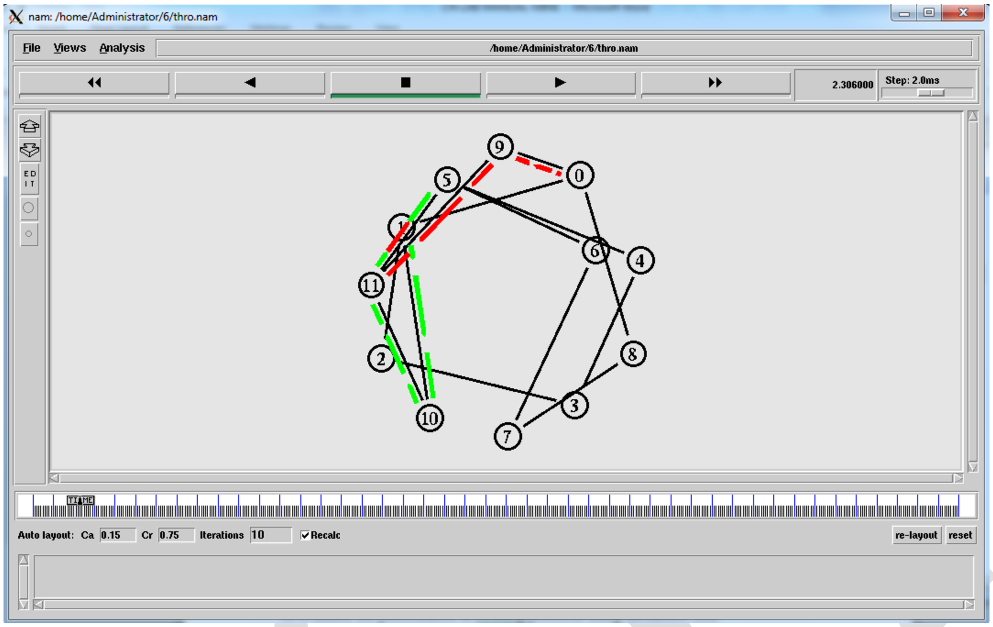
Step 1. Create a Simulator object.
Step 2. Set the routing protocol to Link State.
Step 3. Open the trace and nam trace files.
Step 4. Define the finish procedure.
Step 5. Create nodes and the links between them.
Step 6. Create the agents and attach them to the nodes.
Step 7. Create the applications and attach them to the udp agent.
Step 8. Connect udp and null.
Step 9. Schedule the events
Step 10. Observe the traffic route when link is up and down
Step 11. Run the simulation.

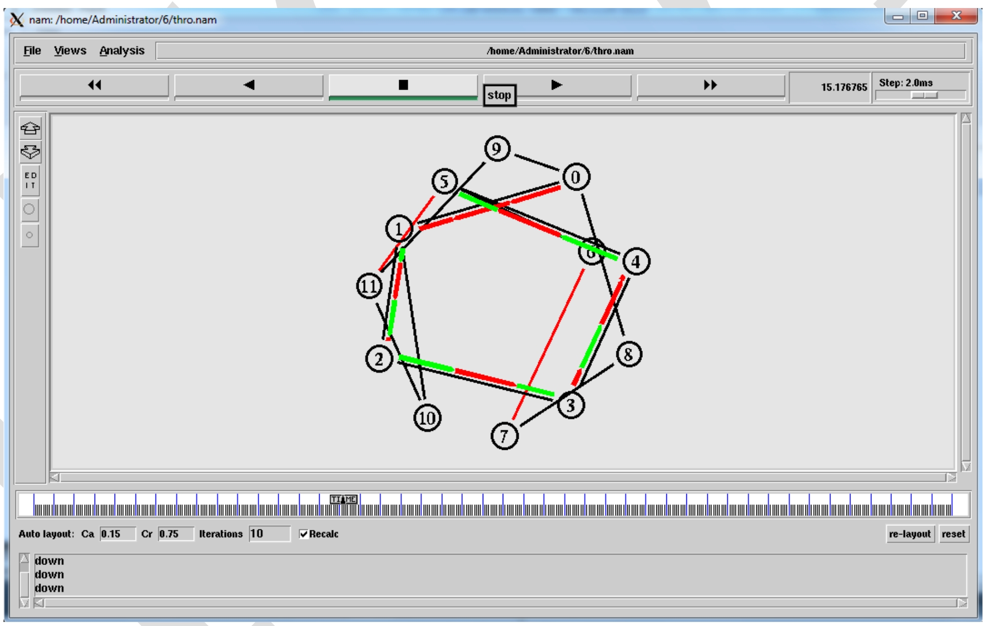
Program:

```
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:





Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus the tcl script to simulate Link State Routing algorithm has been written and executed successfully.

Ex. No: 10	Simulation of Error Correction Codes (CRC)
Date:	

Aim:

To write a java program to simulate error correction Using CRC (Cyclic redundancy check).

Algorithm:

Step 1. Start the program
Step 2. Create an object for the class process.
Step 3. Get the number of bits to be send.
Step 4. Get the divisor value.
Step 5. Generate the dividend value with appending the value of (n-1) number of bits of divisor.
Step 6. Make a change in the message (bits) in sender side.
Step 7. Generate a result whether there is error or not in the message
Step 8. Stop the program.

Program:

```
import java.io.*;
import java.util.*;
class crc_gen
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int[] data;
int[] div;
int[] divisor;
int[] rem;
int[] crc;
int data_bits, divisor_bits, tot_length;
System.out.println("Enter number of data bits : ");
data_bits=Integer.parseInt(br.readLine());
data=new int[data_bits];
System.out.println("Enter data bits : ");
for(int i=0; i<data_bits; i++)
data[i]=Integer.parseInt(br.readLine());
System.out.println("Enter number of bits in divisor : ");
divisor_bits=Integer.parseInt(br.readLine());
divisor=new int[divisor_bits];
System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
divisor[i]=Integer.parseInt(br.readLine());
tot_length=data_bits+divisor_bits-1;
```

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

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

System.out.println();
System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(int i=0; i<crc.length; i++)
crc[i]=Integer.parseInt(br.readLine());
for(int j=0; j<crc.length; j++){
rem[j] = crc[j];
}
rem=divide(crc, divisor, rem);
for(int i=0; i< rem.length; i++)
{
if(rem[i]!=0)
{
System.out.println("Error");
break;
}
if(i==rem.length-1)
System.out.println("No Error");
}
System.out.println("THANK YOU.... :)");
```

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

Output:

Command Prompt

```
D:\CN LAB EXP\CRC>java crc_gen
Enter number of data bits :
7
Enter data bits :
1
0
1
1
0
1
0
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Dividend (after appending 0's) are : 101101000

CRC code :
101101000
Enter CRC code of 9 bits :
1
0
1
1
0
1
0
0
0
No Error
THANK YOU.... :)
```

```
Command Prompt
D:\CN LAB EXP\CRC>java crc_gen
Enter number of data bits :
7
Enter data bits :
1
0
1
1
0
1
0
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Dividend (after appending 0's) are : 101101000

CRC code :
101101000
Enter CRC code of 9 bits :
1
1
1
1
0
1
0
0
0
Error
THANK YOU.... :)
```

Department of CSE		
Performance	30	
Observation	30	
Record	40	
Total	100	

Result:

Thus the java program to simulate error correction using CRC was executed successfully.