TABLE OF CONTENTS

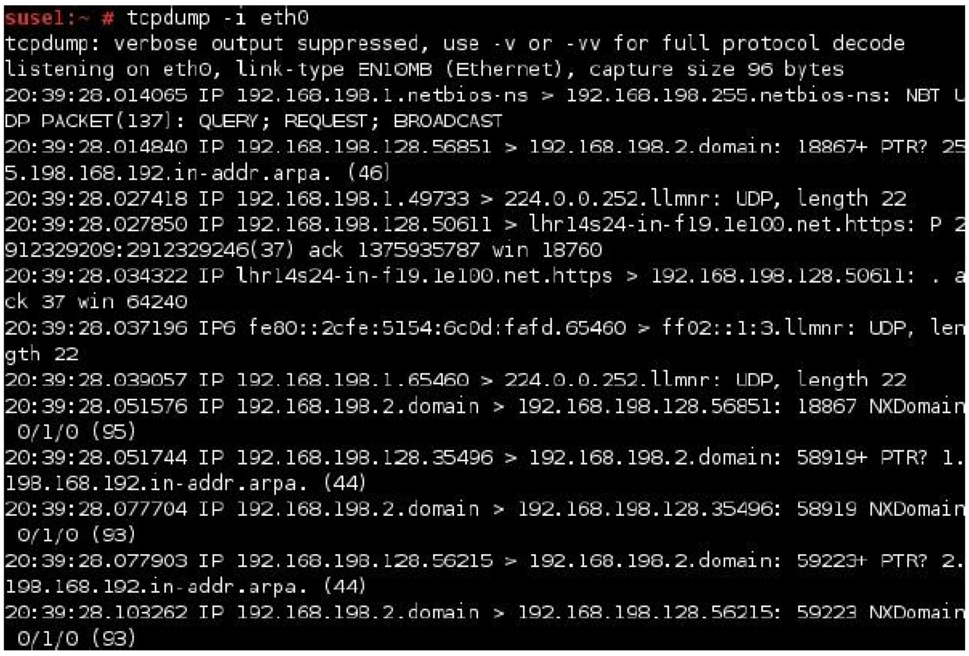
EX NO. 1. NETWORKING COMMANDS

AIM:
To study the basic networking commands.

NETWORKING COMMANDS:

tcpdump

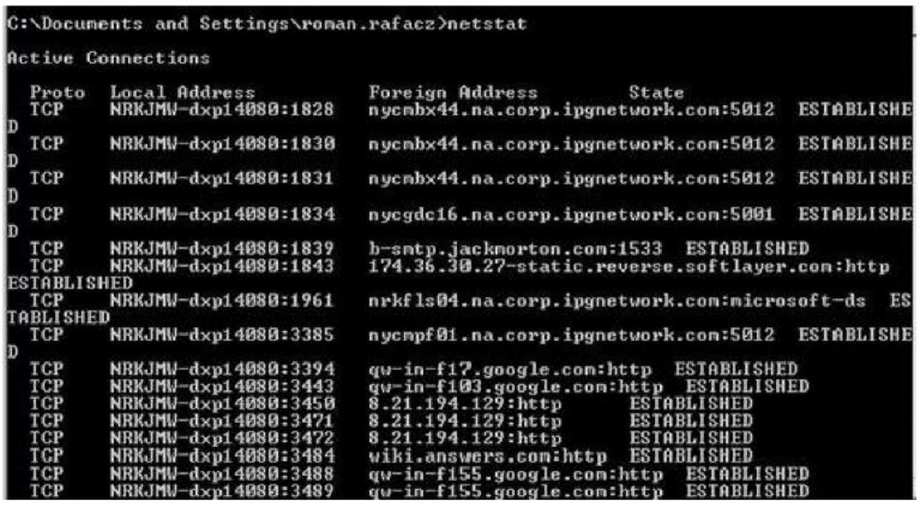
tcpdump is a most powerful and widely used command-line packets sniffer or package analyzer tool which is used to capture or filter TCP/IP packets that received or transferred over a network on a specific interface for analysis.



```
susel:~ # tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
20:39:28.014065 IP 192.168.198.1.netbios-ns > 192.168.198.255.netbios-ns: NBT U
DP PACKET(137): QUERY; REQUEST; BROADCAST
20:39:28.014840 IP 192.168.198.128.56851 > 192.168.198.2.domain: 18867+ PTR? 25
5.198.168.192.in-addr.arpa. (46)
20:39:28.027418 IP 192.168.198.1.49733 > 224.0.0.252.llmnr: UDP, length 22
20:39:28.027850 IP 192.168.198.128.50611 > lhr14s24-in-f19.1e100.net.https: P 2
912329209:2912329246(37) ack 1375935787 win 18760
20:39:28.034322 IP lhr14s24-in-f19.1e100.net.https > 192.168.198.128.50611: . a
ck 37 win 64240
20:39:28.037196 IP6 fe80::2cfe:5154:6c0d:fafd.65460 > ff02::1:3.llmnr: UDP, len
gth 22
20:39:28.039057 IP 192.168.198.1.65460 > 224.0.0.252.llmnr: UDP, length 22
20:39:28.051576 IP 192.168.198.2.domain > 192.168.198.128.56851: 18867 NXDomain
 0/1/0 (95)
20:39:28.051744 IP 192.168.198.128.35496 > 192.168.198.2.domain: 58919+ PTR? 1.
198.168.192.in-addr.arpa. (44)
20:39:28.077704 IP 192.168.198.2.domain > 192.168.198.128.35496: 58919 NXDomain
 0/1/0 (93)
20:39:28.077903 IP 192.168.198.128.56215 > 192.168.198.2.domain: 59223+ PTR? 2.
198.168.192.in-addr.arpa. (44)
20:39:28.103262 IP 192.168.198.2.domain > 192.168.198.128.56215: 59223 NXDomain
 0/1/0 (93)
```

netstat

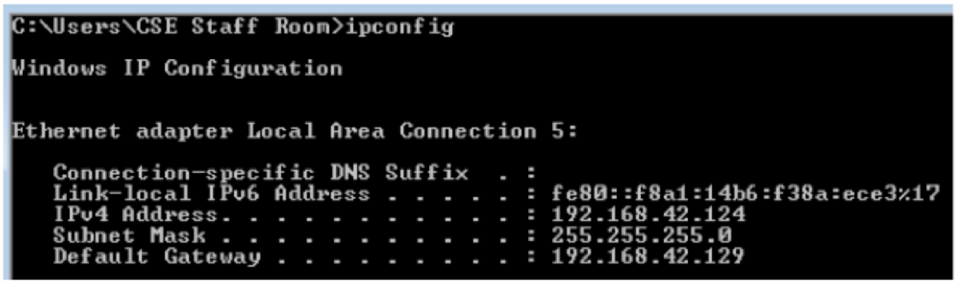
Displays active TCP connections, ports on which the computer is listening, Ethernet statistics, IP routing table, IPv4 statistics and IPv6 statistics. It indicates state of a TCP connection. it's a helpful tool in finding problems and determining the amount of traffic on the network as a performance measurement.



```
C:\Documents and Settings\roman.rafacz>netstat

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    NRKJMW-dxp14080:1828   nycmbx44.na.corp.ipgnetwork.com:5012  ESTABLISHE
D
  TCP    NRKJMW-dxp14080:1830   nycmbx44.na.corp.ipgnetwork.com:5012  ESTABLISHE
D
  TCP    NRKJMW-dxp14080:1831   nycmbx44.na.corp.ipgnetwork.com:5012  ESTABLISHE
D
  TCP    NRKJMW-dxp14080:1834   nycgdc16.na.corp.ipgnetwork.com:5001  ESTABLISHE
D
  TCP    NRKJMW-dxp14080:1839   b-smtp.jackmorton.com:1533  ESTABLISHED
  TCP    NRKJMW-dxp14080:1843   174.36.30.27-static.reverse.softlayer.com:http
ESTABLISHED
  TCP    NRKJMW-dxp14080:1961   nrkfls04.na.corp.ipgnetwork.com:microsoft-ds  ES
TABLISHED
  TCP    NRKJMW-dxp14080:3385   nycmpf01.na.corp.ipgnetwork.com:5012  ESTABLISHE
D
  TCP    NRKJMW-dxp14080:3394   qw-in-f17.google.com:http  ESTABLISHED
  TCP    NRKJMW-dxp14080:3443   qw-in-f103.google.com:http  ESTABLISHED
  TCP    NRKJMW-dxp14080:3450   8.21.194.129:http      ESTABLISHED
  TCP    NRKJMW-dxp14080:3471   8.21.194.129:http      ESTABLISHED
  TCP    NRKJMW-dxp14080:3472   8.21.194.129:http      ESTABLISHED
  TCP    NRKJMW-dxp14080:3484   wiki.answers.com:http  ESTABLISHED
  TCP    NRKJMW-dxp14080:3488   qw-in-f155.google.com:http  ESTABLISHED
  TCP    NRKJMW-dxp14080:3489   qw-in-f155.google.com:http  ESTABLISHED
```

ifconfig / ipconfig

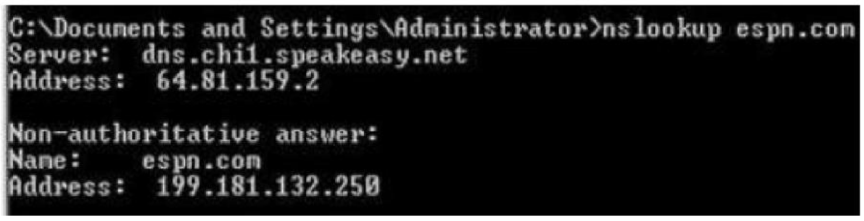
Displays basic current TCP/IP network configuration. It is very useful to troubleshoot networking problems. ipconfig/all is used to provide detailed information such as IP address, subnet mask, MAC address, DNS server, DHCP server, default gateway etc. ipconfig/renew is used to renew a DHCP assigned IP address whereas ipconfig/release is used to discard the assigned DHCP IP address.



```
C:\Users\CSE Staff Room>ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection 5:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::f8a1:14b6:f38a:ece3%17
   IPv4 Address. . . . . . . . . . . : 192.168.42.124
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.42.129
```

nslookup

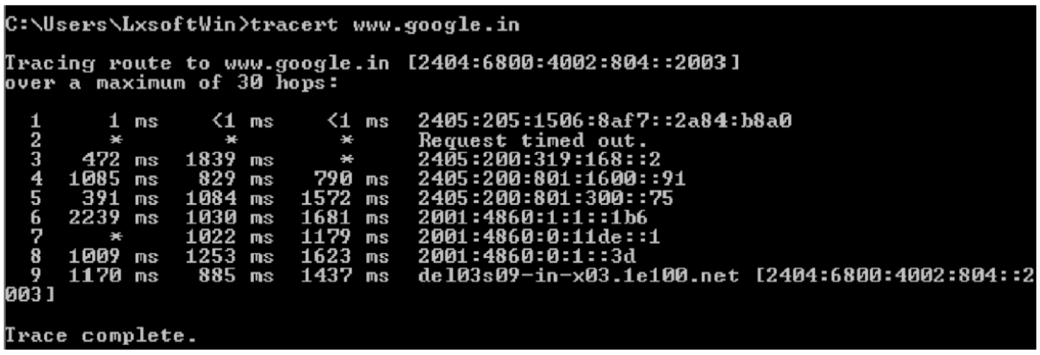
It provides a command-line utility for querying DNS table of a DNS Server. It returns IP address for the given host name.



```
C:\Documents and Settings\Administrator>nslookup espn.com
Server:  dns.chi1.speakeasy.net
Address:  64.81.159.2

Non-authoritative answer:
Name:    espn.com
Address:  199.181.132.250
```

traceroute / tracert

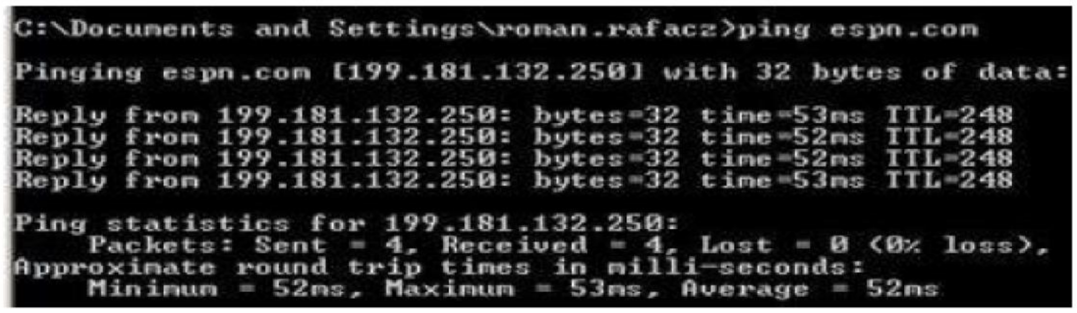
Displays the path taken to a destination by sending ICMP Echo Request messages to the destination with TTL field values. The path displayed is the list of nearest router interfaces taken along each hop in the path between source host and destination.



```
C:\Users\LxsoftWin>tracert www.google.in

Tracing route to www.google.in [2404:6800:4002:804::2003]
over a maximum of 30 hops:

  1     1 ms    <1 ms    <1 ms  2405:205:1506:8af7::2a84:b8a0
  2     *        *        *     Request timed out.
  3   472 ms  1839 ms     *     2405:200:319:168::2
  4  1085 ms   829 ms   790 ms  2405:200:801:1600::91
  5   391 ms  1084 ms  1572 ms  2405:200:801:300::75
  6  2239 ms  1030 ms  1681 ms  2001:4860:1:1::1b6
  7     *     1022 ms  1179 ms  2001:4860:0:11de::1
  8  1009 ms  1253 ms  1623 ms  2001:4860:0:1::3d
  9  1170 ms   885 ms  1437 ms  del03s09-in-x03.1e100.net [2404:6800:4002:804::2
003]

Trace complete.
```

ping

Verifies IP-level connectivity to another TCP/IP computer by sending Internet Control Message Protocol (ICMP) Echo Request messages. The receipt of corresponding Echo Reply messages are displayed, along with round-trip times. Ping is the primary TCP/IP command used to troubleshoot connectivity, reachability, and name resolution.



```
C:\Documents and Settings\roman.rafacz>ping espn.com

Pinging espn.com [199.181.132.250] with 32 bytes of data:

Reply from 199.181.132.250: bytes=32 time=53ms TTL=248
Reply from 199.181.132.250: bytes=32 time=52ms TTL=248
Reply from 199.181.132.250: bytes=32 time=52ms TTL=248
Reply from 199.181.132.250: bytes=32 time=53ms TTL=248

Ping statistics for 199.181.132.250:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 52ms, Maximum = 53ms, Average = 52ms
```

RESULT:

Thus the TCP/IP network command utilities is studied in detail.

EX.NO. 2. HTTP WEB CLIENT PROGRAM

AIM:

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

ALGORITHM:

Step 1: Start the program.
Step 2: Set a server port as 80.
Step 3: Using HTTP services create a Socket for server by specifying the server port.
Step 4: Use HTTP socket for connecting the client to the URL.
Step 5: Use BufferedReader to output stream to place the response from the server by the client.
Step 6: Close the Connection as soon the request is been serviced. Use Malformed URL exception If any errors in grabbing the server.
Step 7: Stop the program.

PROGRAM:

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

public class SocketHTTPClient {
public static void main(String[] args) {
      try {
         // Create a socket to connect to the website
         Socket socket = new Socket("www.martinbroadhurst.com", 80);

         // Get the output stream to send request
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

         // Send an HTTP GET request
         out.println("GET / HTTP/1.1");
         out.println("Host: www.martinbroadhurst.com");
         out.println("");  // End of the HTTP request

         // Get the input stream to read the server's response
         BufferedReader          in          =          new          BufferedReader(new
InputStreamReader(socket.getInputStream()));

         // Read and print the server's response
         String responseLine;
         while ((responseLine = in.readLine()) != null) {
            System.out.println(responseLine);
         }

         // Close the streams and socket
         in.close();
         out.close();
         socket.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

EXECUTION:

Step 1: Save SocketHTTPClient java file as SocketHTTPClient.java.

Step 2: Open two cmd prompt.

Step 3: Set path for javac compiler and java interpreter in cmd prompt as follows

```
set path=” C:\Program Files\Java\jdk1.7.0_79\bin”;
```

Step 4: Compile your java program as follows

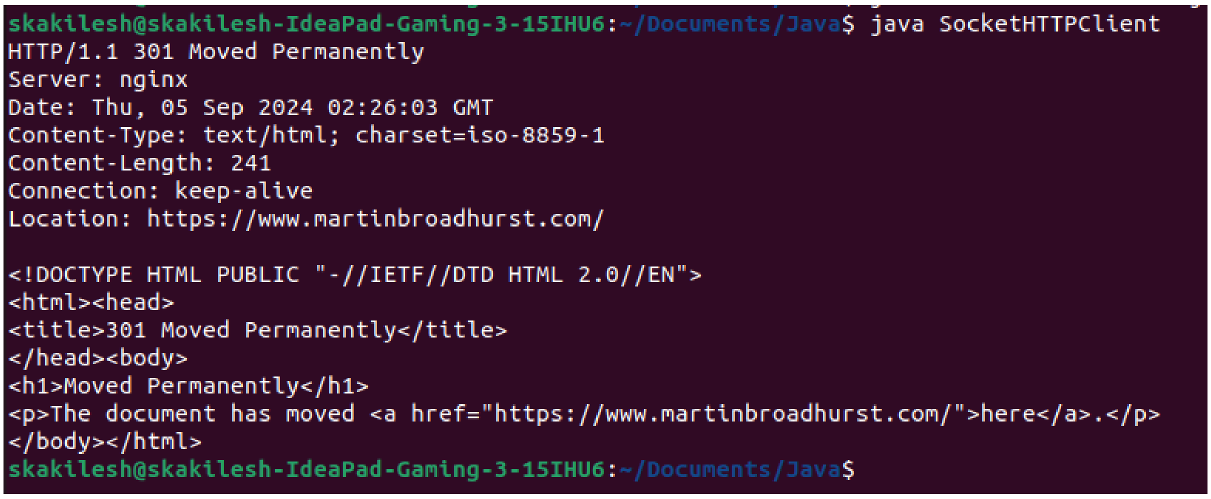
```
javac    SocketHTTPClient.java
```

It generates Bytecode file (or Object file) as SocketHTTPClient.class

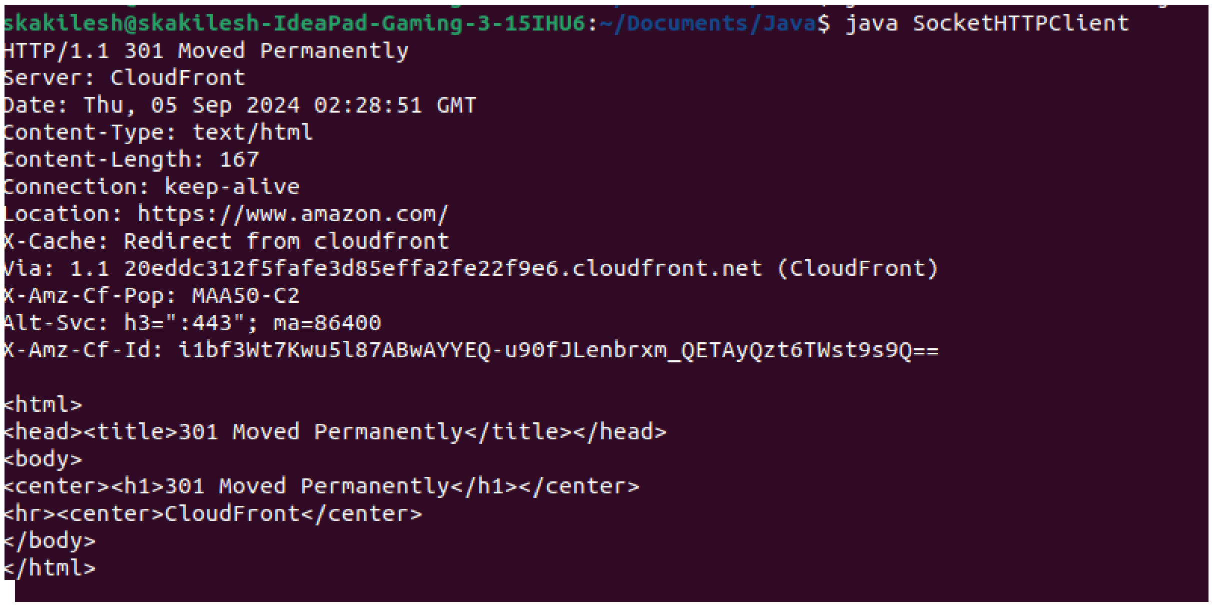
Step 5: Execute your java program as follows

```
java SocketHTTPClient
```

OUTPUT:



```
skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java SocketHTTPClient
HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Thu, 05 Sep 2024 02:26:03 GMT
Content-Type: text/html; charset=iso-8859-1
Content-Length: 241
Connection: keep-alive
Location: https://www.martinbroadhurst.com/

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="https://www.martinbroadhurst.com/">here</a>.</p>
</body></html>
skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$
```



```
skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java SocketHTTPClient
HTTP/1.1 301 Moved Permanently
Server: CloudFront
Date: Thu, 05 Sep 2024 02:28:51 GMT
Content-Type: text/html
Content-Length: 167
Connection: keep-alive
Location: https://www.amazon.com/
X-Cache: Redirect from cloudfront
Via: 1.1 20eddc312f5fafe3d85effa2fe22f9e6.cloudfront.net (CloudFront)
X-Amz-Cf-Pop: MAA50-C2
Alt-Svc: h3=":443"; ma=86400
X-Amz-Cf-Id: i1bf3Wt7Kwu5l87ABwAYYEQ-u90fJLenbrxm_QETAyQzt6TWst9s9Q==

<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>CloudFront</center>
</body>
</html>
```

RESULT:

Thus the java HTTP web client program to download a web page using TCP sockets has been executed successfully and output got verified.

EX.NO. 3A: ECHO CLIENT AND ECHO SERVER APPLICATIONS USING TCP SOCKETS

AIM:

To write a socket program to implement echo client and echo server applications using TCP sockets in java.

ALGORITHM:

ECHO SERVER:

Step 1: Start the program.
Step 2: Import necessary packages to access the predefined classes and its methods.
Step 3: Create server side socket using ServerSocket class.
Step 4: Accept connection from client by using accept() method.
Step 5: Initialize object for DataInputStream class with getInputStream() method of Socket class to receive message from client.
Step 6: Initialize object for DataOutputStream class with getOutputStream() method of Socket class to send message to client.
Step 7: Using readUTF() method receive message from client.
Step 8: Using writeUTF() method to echo received message to client.
Step 8: Stop the program.

ECHO CLIENT:

Step 1: Start the program.
Step 2: Import necessary packages to access the predefined classes and its methods.
Step 3: Create client side socket using Socket class.
Step 4: Initialize object for DataInputStream class with getInputStream() method of Socket class to receive message from client.
Step 5: Initialize object for DataOutputStream class with getOutputStream() method of Socket class to send message to client.
Step 6: Using readUTF() and writeUTF() method send and receive mesaage from server.
Step 7: Stop the program.

Code:

EchoServer

```
import java.io.*;
import java.net.*;
import java.util.*;
public class ChatServer
{
public static void main(String args[])
{
try
{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new
DataOutputStream(s.getOutputStream());
Scanner input=new Scanner(System.in);
String senddata="";
String recievedata="";
while(!recievedata.equals("stop"))
{
recievedata=din.readUTF();System.out.println("CLIENT SAYS :"+recievedata);
System.out.print("TO CLIENT :");
senddata=input.nextLine();
dout.writeUTF(senddata);
}
din.close();
dout.close();
s.close();
ss.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

EchoClient

```
import java.io.*;
import java.net.*;
import java.util.*;
public class ChatClient
{
public static void main(String args[])
{
try
{
Socket s=new Socket("localhost",6666);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new
DataOutputStream(s.getOutputStream());
Scanner input=new Scanner(System.in);
String senddata="";
String recievedata="";
while(!senddata.equals("stop"))
{
System.out.print("TO SERVER :");
senddata=input.nextLine();
dout.writeUTF(senddata);recievedata=din.readUTF();
System.out.println("SERVER SAYS :"+recievedata);
}
din.close();
dout.close();
s.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

EXECUTION:

Step 1: Save Echo Server java file as EchoServer.java and Echo Client java file as EchoClient.java.
Step 2: Open two cmd prompt for client and server.
Step 3: Set path for javac compiler and java interpreter in cmd prompt as follows

```
set path=" C:\Program Files\Java\jdk1.7.0_79\bin";
```

Step 4: Compile your server java program as follows

```
javac   EchoServer.java
```

It generates Bytecode file (or Object file) as EchoServer.class

Step 5: Compile your client java program as follows

```
javac   EchoClient.java
```

It generates Bytecode file (or Object file) as EchoClient.class

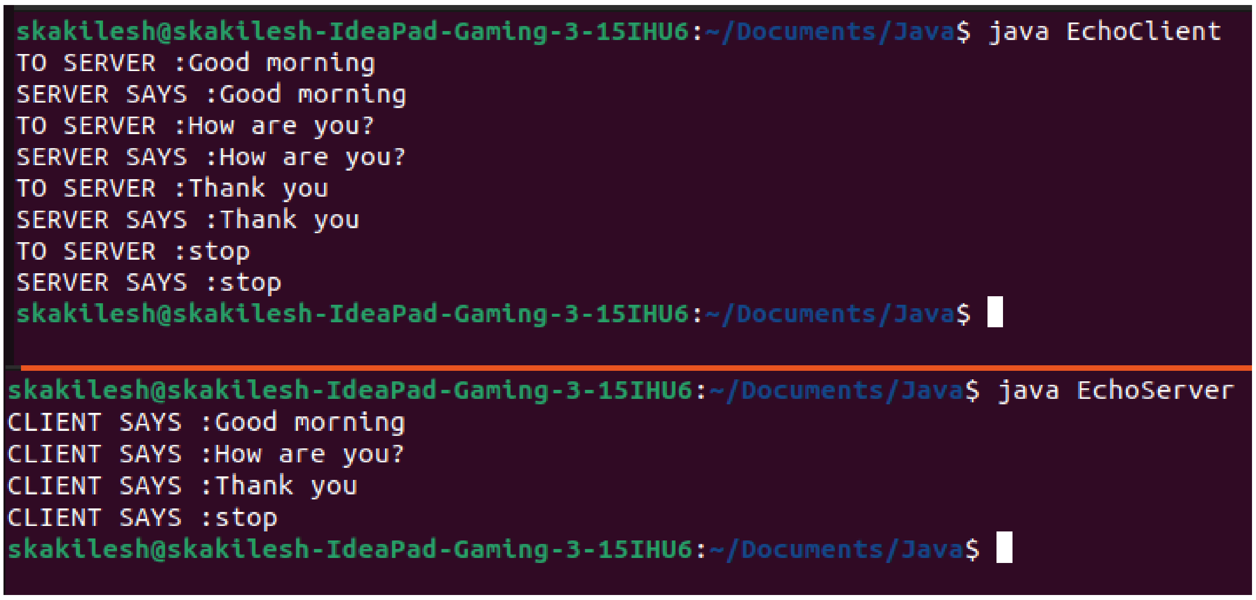
Step 6: Execute your server java program as follows

```
java EchoServer
```

Step 7: Execute your client java program as follows

```
java EchoClient
```

OUTPUT:



```
skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java EchoClient
TO SERVER :Good morning
SERVER SAYS :Good morning
TO SERVER :How are you?
SERVER SAYS :How are you?
TO SERVER :Thank you
SERVER SAYS :Thank you
TO SERVER :stop
SERVER SAYS :stop
skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$
```

```
skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java EchoServer
CLIENT SAYS :Good morning
CLIENT SAYS :How are you?
CLIENT SAYS :Thank you
CLIENT SAYS :stop
skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$
```

RESULT:

Thus the socket program to implement echo client and echo server applications using TCP sockets in java has been executed successfully and output got verified.

EX.NO. 3B: CHAT APPLICATION USING TCP SOCKETS

AIM:

To write a socket program to implement chat application using TCP sockets in java.

ALGORITHM:

CHAT SERVER:
Step 1: Start the program.
Step 2: Import necessary packages to access the predefined classes and its methods.
Step 3: Create server side socket using ServerSocket class.
Step 4: Accept connection from client by using accept() method.
Step 5: Initialize object for DataInputStream class with getInputStream() method of Socket class to receive message from client.
Step 6: Initialize object for DataOutputStream class with getOutputStream() method of Socket class to send message to client.
Step 7: Using readUTF() and writeUTF() method send and receive mesaage between client and server.
Step 8: Stop the program.

CHAT CLIENT:

Step 1: Start the program.
Step 2: Import necessary packages to access the predefined classes and its methods.
Step 3: Create client side socket using Socket class.
Step 4: Initialize object for DataInputStream class with getInputStream() method of Socket class to receive message from client.
Step 5: Initialize object for DataOutputStream class with getOutputStream() method of Socket class to send message to client.
Step 6: Using readUTF() and writeUTF() method send and receive mesaage between client and server.
Step 7: Stop the program.
Code

ChatClient.java:

```
import java.io.*;
import java.net.*;
import java.util.*;
public class ChatClient
{
public static void main(String args[])
{
try
{
Socket s=new Socket("localhost",6666);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new
DataOutputStream(s.getOutputStream());
Scanner input=new Scanner(System.in);
String senddata="";
String recievedata="";
while(!senddata.equals("stop"))
{
System.out.print("TO SERVER :");
senddata=input.nextLine();
dout.writeUTF(senddata);recievedata=din.readUTF();
System.out.println("SERVER SAYS :"+recievedata);
}
din.close();
dout.close();
s.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

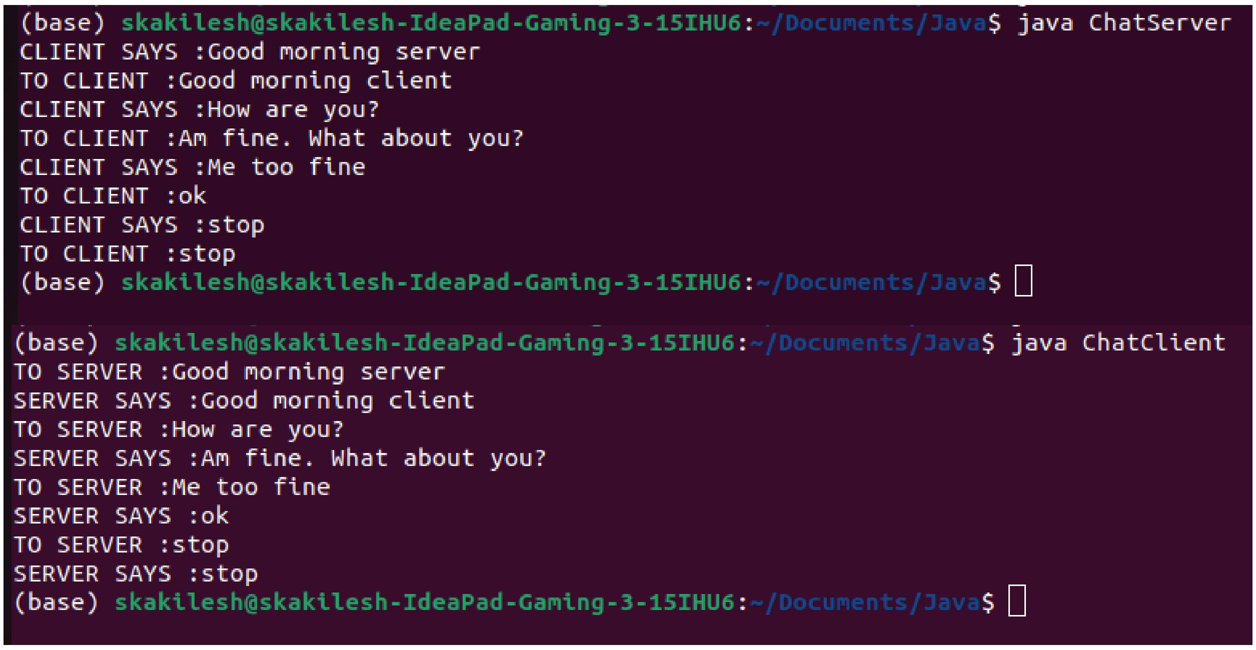
ChatServer.java:

```
import java.io.*;
import java.net.*;
import java.util.*;
public class ChatServer
{
public static void main(String args[])
{
try
{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new
DataOutputStream(s.getOutputStream());
Scanner input=new Scanner(System.in);
String senddata="";
String recievedata="";
while(!recievedata.equals("stop"))
{
recievedata=din.readUTF();System.out.println("CLIENT SAYS :"+recievedata);
System.out.print("TO CLIENT :");
senddata=input.nextLine();
dout.writeUTF(senddata);
}
din.close();
dout.close();
s.close();
ss.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

EXECUTION:

Step 1: Save Chat Server java file as ChatServer.java and Chat Client java file as ChatClient.java.
Step 2: Open two cmd prompt for client and server.
Step 3: Set path for javac compiler and java interpreter in cmd prompt as follows
set path=" C:\Program Files\Java\jdk1.7.0_79\bin";
Step 4: Compile your server java program as follows
javac ChatServer.java
It generates Bytecode file (or Object file) as ChatServer.class
Step 5: Compile your client java program as follows
javac ChatClient.java
It generates Bytecode file (or Object file) as ChatClient.class
Step 6: Execute your server java program as follows
java ChatServer
Step 7: Execute your client java program as follows
java ChatClient

OUTPUT:



```
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java ChatServer
CLIENT SAYS :Good morning server
TO CLIENT :Good morning client
CLIENT SAYS :How are you?
TO CLIENT :Am fine. What about you?
CLIENT SAYS :Me too fine
TO CLIENT :ok
CLIENT SAYS :stop
TO CLIENT :stop
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$
```

```
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java ChatClient
TO SERVER :Good morning server
SERVER SAYS :Good morning client
TO SERVER :How are you?
SERVER SAYS :Am fine. What about you?
TO SERVER :Me too fine
SERVER SAYS :ok
TO SERVER :stop
SERVER SAYS :stop
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$
```

RESULT:

Thus the socket program to implement chat application using TCP sockets in java has been executed successfully and output got verified.

EX.NO. 3C: FILE TRANSFER APPLICATION USING TCP SOCKETS

AIM:

To write a socket program to implement file transfer application using TCP sockets in java.

ALGORITHM:

FTP SERVER:

Step 1: Start the program.
Step 2: Import necessary packages to access the predefined classes and its methods.
Step 3: Create server side socket using ServerSocket class.
Step 4: Accept connection from client by using accept() method.
Step 5: Initialize PrintStream class object with getOutputStream() method to send content of a file from server to client.
Step 6: Using File class select a file to send and use buffer as a internal storage.
Step 7: Stop the program.

FTP CLIENT:

Step 1: Start the program.
Step 2: Import necessary packages to access the predefined classes and its methods.
Step 3: Create client side socket using Socket class.
Step 4: Initialize InputStreamReader class object with getInputStream() method to receive content of a file from server.
Step 5: Using File class create a file to receive content from server and use buffer as a internal storage.
Step 6: Stop the program.

Code

FTPServer.java

```
import java.io.*;
import java.net.*;
public class FTPServer
{
public static void main(String args[]) throws Exception
{
ServerSocket ss=new ServerSocket(1024);
System.out.println("ServerSocket Generated");
Socket s=ss.accept();
System.out.println("ServerSocket Accepted");
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
PrintStream p=new PrintStream(s.getOutputStream());
String fname,str;
System.out.println("Enter a File Name:");
fname=br.readLine();
File f1=new File(fname);
if(f1.exists())
{BufferedReader br1=new BufferedReader(new
FileReader(fname));
while((str=br1.readLine())!=null)
p.println(str);
}
p.close();
}
}
```

FTPClient.java

```
import java.io.*;
import java.net.*;
public class FTPClient
{
public static void main(String asd[]) throws Exception
{
InetAddress ia=InetAddress.getLocalHost();
Socket s=new Socket(ia,1024);
String fname,str;
System.out.println("Enter a new File Name:");
BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
fname=br.readLine();
File f1=new File(fname);
PrintWriter p=new PrintWriter(new FileWriter(fname));
BufferedReader br1=new BufferedReader(new
InputStreamReader(s.getInputStream()));
while((str=br1.readLine())!=null)
p.println(str);
p.close();
s.close();
}
}
```

EXECUTION:

Step 1: Save FTP Server java file as FTPServer.java and FTP Client java file as FTPClient.java.
Step 2: Open two cmd prompt for client and server.
Step 3: Set path for javac compiler and java interpreter in cmd prompt as follows

```
set path=" C:\Program Files\Java\jdk1.7.0_79\bin";
```

Step 4: Compile your server java program as follows

```
javac   FTPServer.java
```

It generates Bytecode file (or Object file) as FTPServer.class
Step 5: Compile your client java program as follows

```
javac   FTPClient.java
```

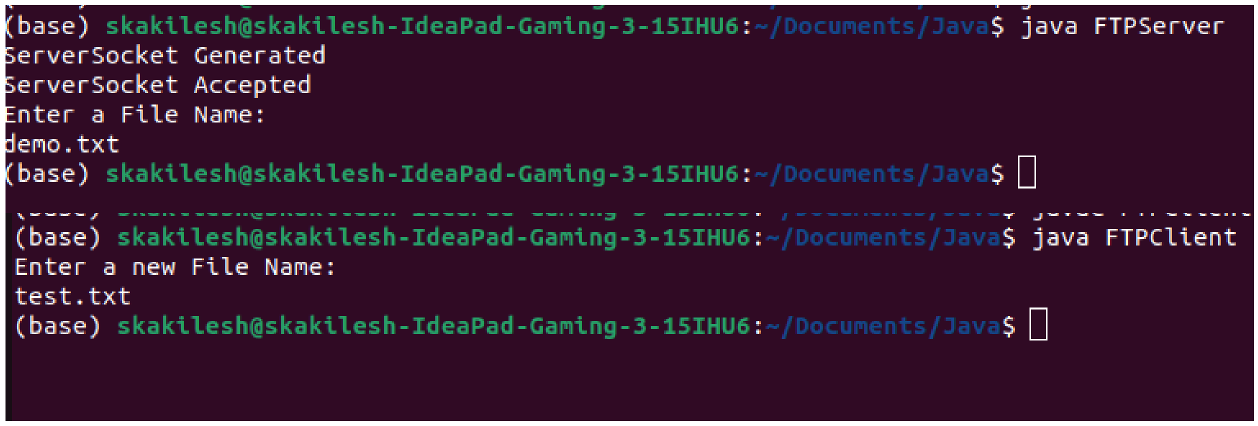
It generates Bytecode file (or Object file) as FTPClient.class
Step 6: Execute your server java program as follows

```
java FTPServer
```

Step 7: Execute your client java program as follows

```
java FTPClient
```

OUTPUT:



```
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java FTPServer
ServerSocket Generated
ServerSocket Accepted
Enter a File Name:
demo.txt
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$
```

```
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java FTPClient
Enter a new File Name:
test.txt
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$
```

RESULT:

Thus the socket program to implement file transfer application using TCP sockets in java has been executed successfully and output got verified.

EX.NO. 4: SIMULATION OF DNS USING UDP SOCKETS

AIM:

To write a socket program for simulating DNS using UDP sockets in java.

ALGORITHM:

CLIENT

Step 1: Start the program
Step 2: Using datagram sockets UDP function is established.
Step 3: Get the server name address to be converted into IP address.
Step 4: Send this server name to server.
Step 5: Server returns the IP address to client.
Step 6: Stop the program.

SERVER

Step 1: Start the program
Step 2: Accept the socket which is created by the client.
Step 3: Server maintains the table in which IP and corresponding server name are stored.
Step 4: Read the server name which is send by the client.
Step 5: Map the server name with its IP address and return the IP address to client.
Step 6: Stop the program.

Code

Server.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Server
{
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();
//System.out.println(s);
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.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Client{
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 SERVER/DOMAIN NAME:");
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 is: "+s.trim());
client.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

EXECUTION:

Step 1: Save Server java file as Server.java and Client java file as Client.java.
Step 2: Open two cmd prompt for client and server.
Step 3: Set path for javac compiler and java interpreter in cmd prompt as follows

```
set path=" C:\Program Files\Java\jdk1.7.0_79\bin";
```

Step 4: Compile your server java program as follows

```
javac   Server.java
```

It generates Bytecode file (or Object file) as Server.class

Step 5: Compile your client java program as follows

```
javac   Client.java
```

It generates Bytecode file (or Object file) as Client.class

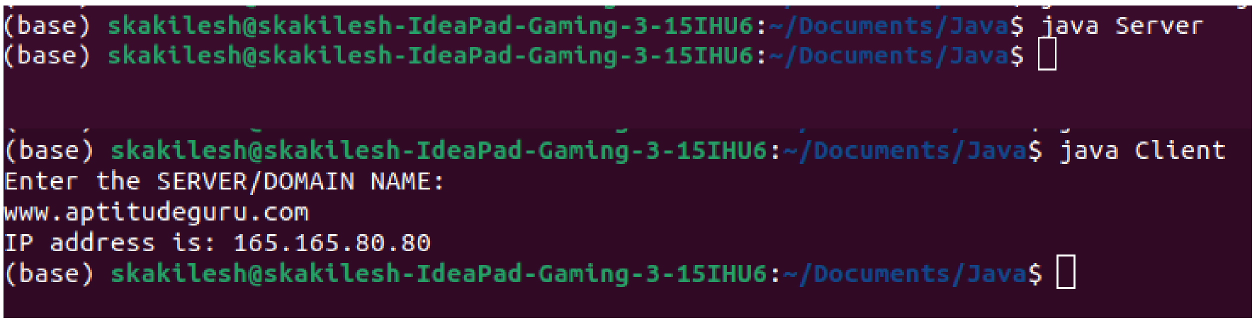
Step 6: Execute your server java program as follows

```
java Server
```

Step 7: Execute your client java program as follows

```
java Client
```

OUTPUT:



```
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java Server
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ 
```

```
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java Client
Enter the SERVER/DOMAIN NAME:
www.aptitudeguru.com
IP address is: 165.165.80.80
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ 
```

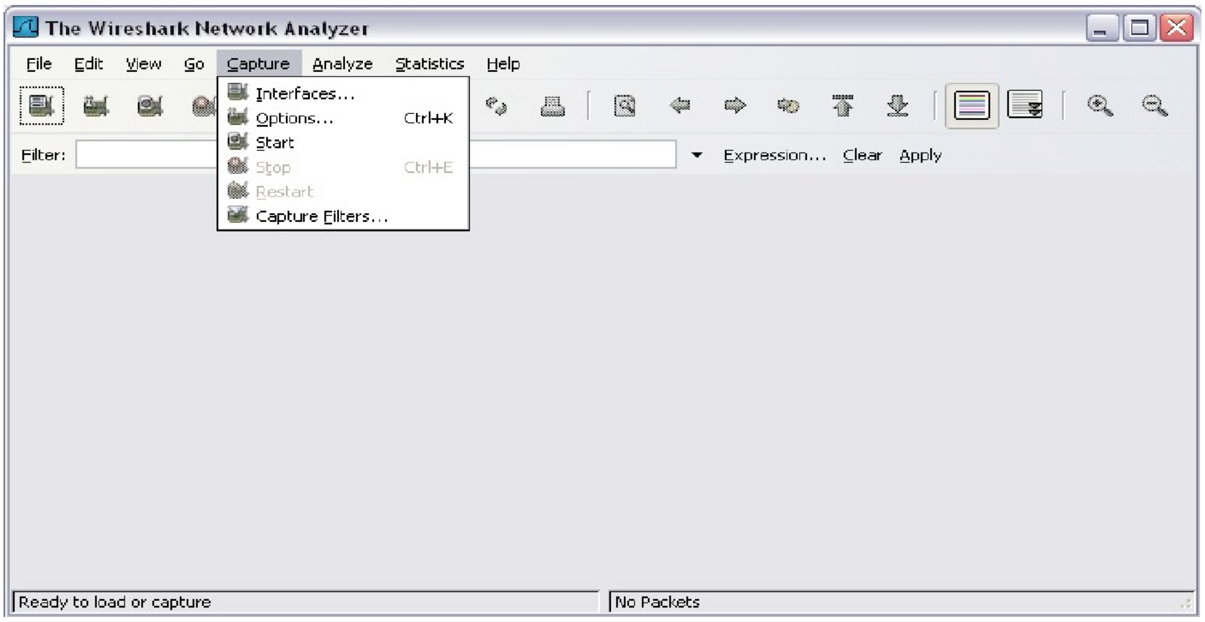
EX.NO. 5. WIRESHARK TOOL TO CAPTURE AND EXAMINE PACKETS

AIM:

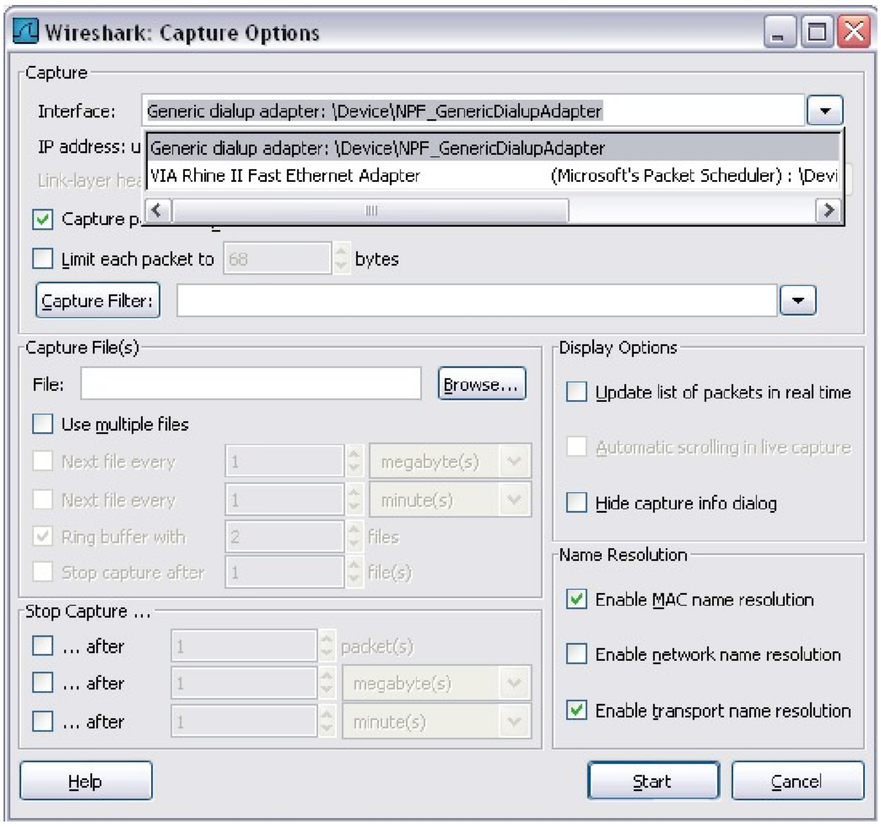
To capture ping and traceroute PDUs using a network protocol analyzer and examine in detail.

WIRESHARK TOOL

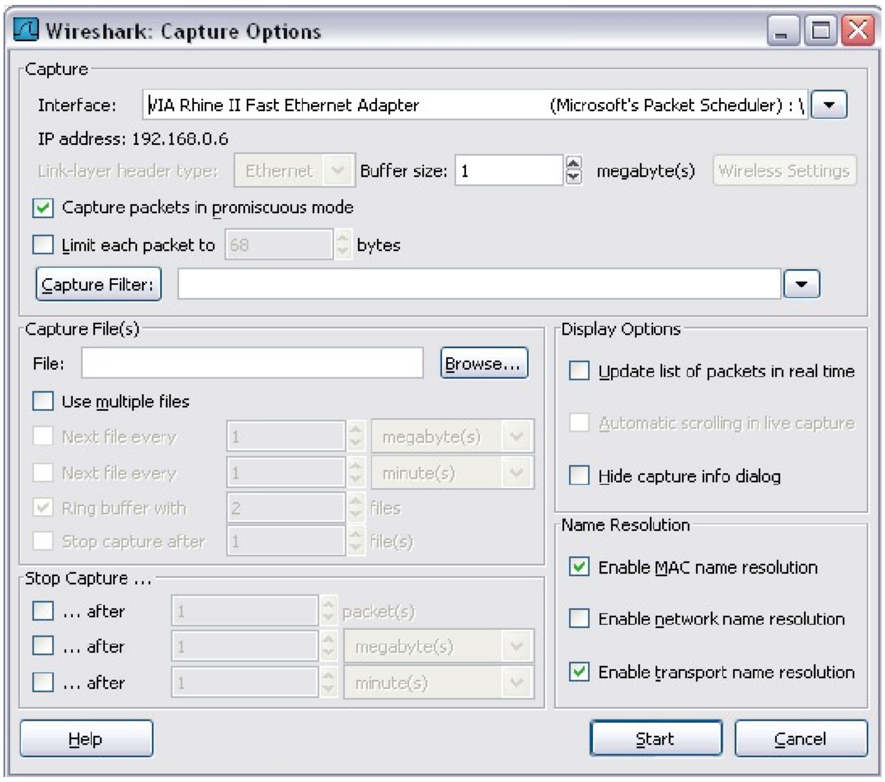
When Wireshark is launched, the screen below is displayed.



To start data capture it is first necessary to go to the **Capture** menu and select the **Options** choice. The **Options** dialog provides a range of settings and filters which determines which and how much data traffic is captured.


Wireshark: Capture Options
Capture
Interface: Generic dialup adapter: \Device\NPF_GenericDialupAdapter
IP address: u
Generic dialup adapter: \Device\NPF_GenericDialupAdapter
VIA Rhine II Fast Ethernet Adapter (Microsoft's Packet Scheduler) : \Devi
Link-layer hea
Capture p
Limit each packet to 68 bytes
Capture Filter:
Capture File(s)
File: Browse...
Use multiple files
Next file every 1 megabyte(s)
Next file every 1 minute(s)
Ring buffer with 2 files
Stop capture after 1 file(s)
Stop Capture ...
... after 1 packet(s)
... after 1 megabyte(s)
... after 1 minute(s)
Display Options
Update list of packets in real time
Automatic scrolling in live capture
Hide capture info dialog
Name Resolution
Enable MAC name resolution
Enable network name resolution
Enable transport name resolution
Help Start Cancel

First, it is necessary to ensure that Wireshark is set to monitor the correct interface. From the **Interface** drop-down list, select the network adapter in use. Typically, for a computer this will be the connected Ethernet Adapter. Then other Options can be set. Among those available in **Capture Options,** the two highlighted below are worth examination.



Wireshark: Capture Options

Capture

Interface: VIA Rhine II Fast Ethernet Adapter (Microsoft's Packet Scheduler) : \

IP address: 192.168.0.6

Link-layer header type: Ethernet Buffer size: 1 megabyte(s) Wireless Settings

[✓] Capture packets in promiscuous mode

[ ] Limit each packet to 68 bytes

Capture Filter:

Capture File(s)

File: Browse...

[ ] Use multiple files

[ ] Next file every 1 megabyte(s)

[ ] Next file every 1 minute(s)

[✓] Ring buffer with 2 files

[ ] Stop capture after 1 file(s)

Stop Capture ...

[ ] ... after 1 packet(s)

[ ] ... after 1 megabyte(s)

[ ] ... after 1 minute(s)

Display Options

[ ] Update list of packets in real time

[ ] Automatic scrolling in live capture

[ ] Hide capture info dialog

Name Resolution

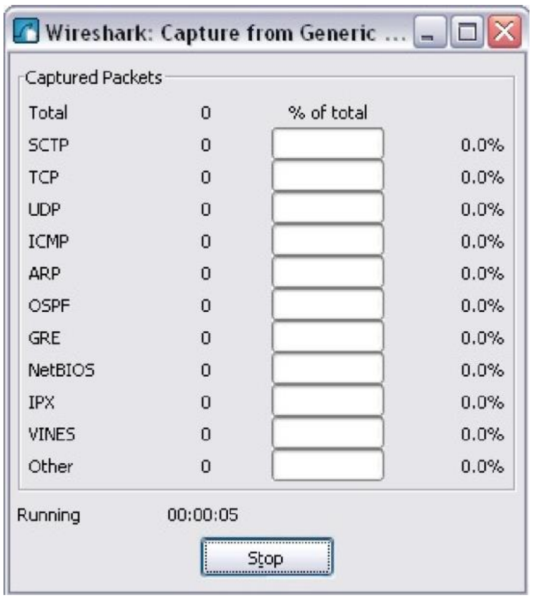
[✓] Enable MAC name resolution

[ ] Enable network name resolution

[✓] Enable transport name resolution

Help Start Cancel

Clicking on the **Start** button starts the data capture process and a message box displays the progress of this process.



Wireshark: Capture from Generic ...

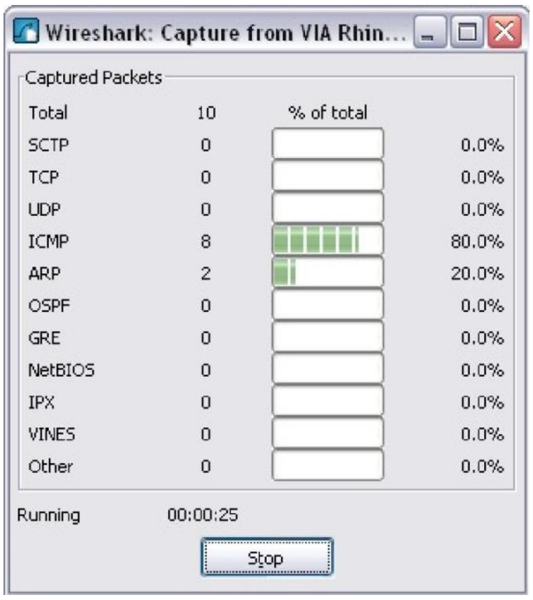
Captured Packets

		% of total	
Total	0		
SCTP	0		0.0%
TCP	0		0.0%
UDP	0		0.0%
ICMP	0		0.0%
ARP	0		0.0%
OSPF	0		0.0%
GRE	0		0.0%
NetBIOS	0		0.0%
IPX	0		0.0%
VINES	0		0.0%
Other	0		0.0%

Running 00:00:05

Stop

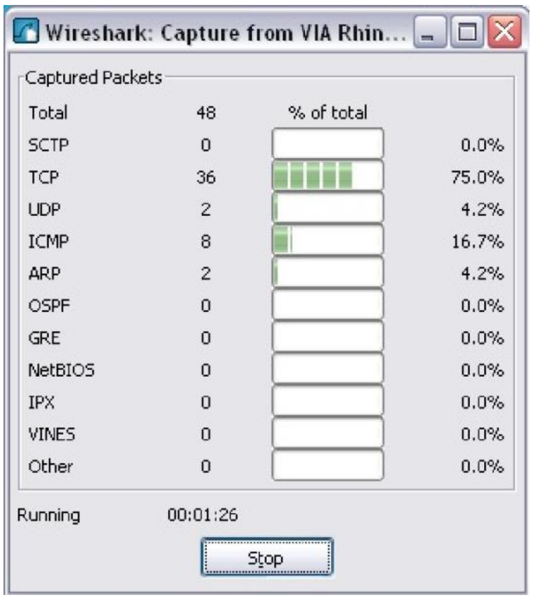
As data PDUs are captured, the types and number are indicated in the message box


Wireshark: Capture from VIA Rhin...

Captured Packets

		% of total	
Total	10		
SCTP	0		0.0%
TCP	0		0.0%
UDP	0		0.0%
ICMP	8		80.0%
ARP	2		20.0%
OSPF	0		0.0%
GRE	0		0.0%
NetBIOS	0		0.0%
IPX	0		0.0%
VINES	0		0.0%
Other	0		0.0%

Running 00:00:25

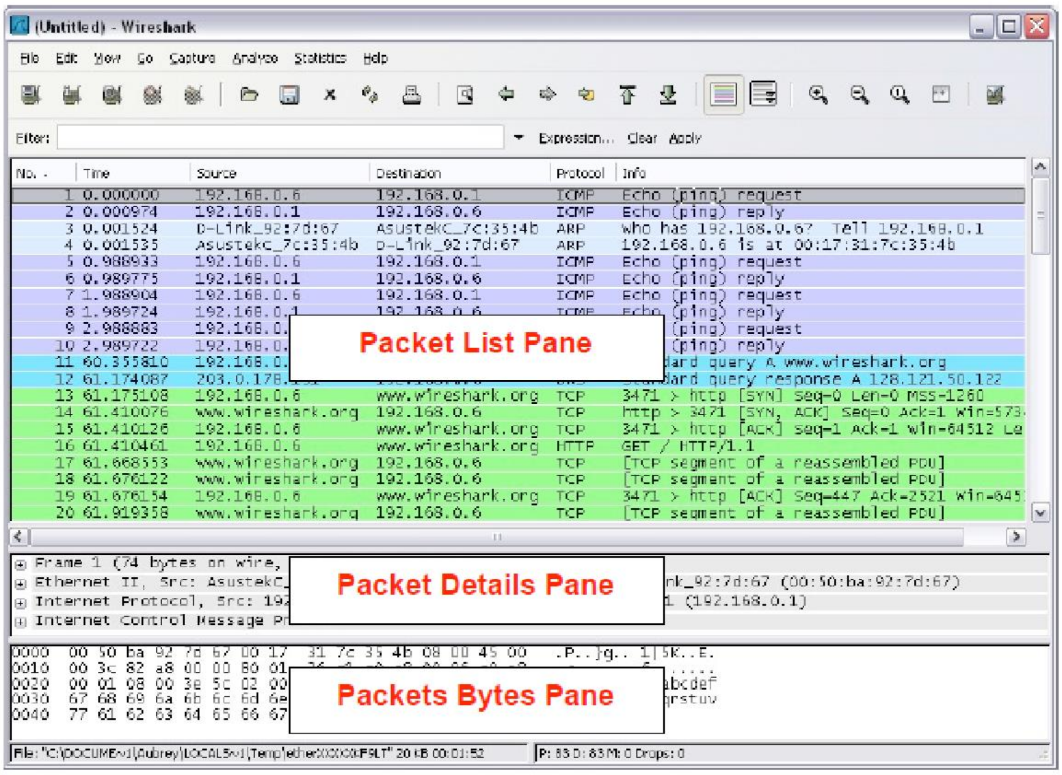

Wireshark: Capture from VIA Rhin...

Captured Packets

		% of total	
Total	48		
SCTP	0		0.0%
TCP	36		75.0%
UDP	2		4.2%
ICMP	8		16.7%
ARP	2		4.2%
OSPF	0		0.0%
GRE	0		0.0%
NetBIOS	0		0.0%
IPX	0		0.0%
VINES	0		0.0%
Other	0		0.0%

Running 00:01:26

The examples above show the capture of a ping process and then accessing a web page. When the **Stop** button is clicked, the capture process is terminated and the main screen is displayed.
This main display window of Wireshark has three panes.


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: Expression... Clear Apply
Packet List Pane
Packet Details Pane
Packets Bytes Pane

```
No. -  Time       Source              Destination         Protocol  Info
  1 0.000000   192.168.0.6         192.168.0.1         ICMP      Echo (ping) request
  2 0.000974   192.168.0.1         192.168.0.6         ICMP      Echo (ping) reply
  3 0.001524   D-Link_92:7d:67     AsustekC_7c:35:4b   ARP       Who has 192.168.0.6?  Tell 192.168.0.1
  4 0.001535   AsustekC_7c:35:4b   D-Link_92:7d:67     ARP       192.168.0.6 is at 00:17:31:7c:35:4b
  5 0.988933   192.168.0.6         192.168.0.1         ICMP      Echo (ping) request
  6 0.989775   192.168.0.1         192.168.0.6         ICMP      Echo (ping) reply
  7 1.988904   192.168.0.6         192.168.0.1         ICMP      Echo (ping) request
  8 1.989724   192.168.0.1         192.168.0.6         ICMP      Echo (ping) reply
  9 2.988883   192.168.0.                                         (ping) request
 10 2.989722   192.168.0.                                         (ping) reply
 11 60.355810  192.168.0.                                        dard query A www.wireshark.org
 12 61.174087  203.0.178.                                        dard query response A 128.121.50.122
 13 61.175108  192.168.0.6         www.wireshark.org   TCP       3471 > http [SYN] Seq=0 Len=0 MSS=1260
 14 61.410076  www.wireshark.org   192.168.0.6         TCP       http > 3471 [SYN, ACK] Seq=0 Ack=1 Win=573
 15 61.410126  192.168.0.6         www.wireshark.org   TCP       3471 > http [ACK] Seq=1 Ack=1 Win=64512 Le
 16 61.410461  192.168.0.6         www.wireshark.org   HTTP      GET / HTTP/1.1
 17 61.668553  www.wireshark.org   192.168.0.6         TCP       [TCP segment of a reassembled PDU]
 18 61.676122  www.wireshark.org   192.168.0.6         TCP       [TCP segment of a reassembled PDU]
 19 61.676154  192.168.0.6         www.wireshark.org   TCP       3471 > http [ACK] Seq=447 Ack=2521 Win=645
 20 61.919358  www.wireshark.org   192.168.0.6         TCP       [TCP segment of a reassembled PDU]
```

```
⊞ Frame 1 (74 bytes on wire,
⊞ Ethernet II, Src: AsustekC_                         nk_92:7d:67 (00:50:ba:92:7d:67)
⊞ Internet Protocol, Src: 192                         1 (192.168.0.1)
⊞ Internet Control Message Pr
```

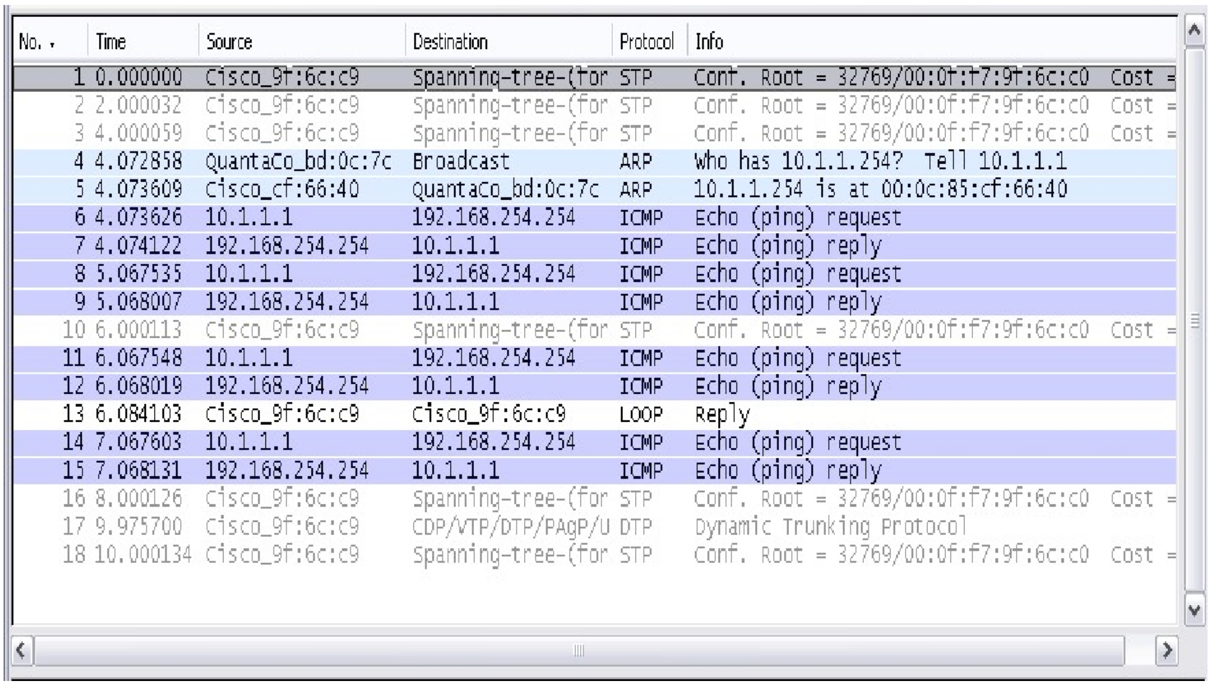
```
0000  00 50 ba 92 7d 67 00 17  31 7c 35 4b 08 00 45 00   .P..}g.. 1|5K..E.
0010  00 3c 82 a8 00 00 80 01
0020  00 01 08 00 3e 5c 02 00                             abcdef
0030  67 68 69 6a 6b 6c 6d 6e                             qrstuv
0040  77 61 62 63 64 65 66 67
```

File: "C:\DOCUME~1\Aubrey\LOCALS~1\Temp\etherXXXXXXF9LT" 20 KB 00:01:52 | P: 83 D: 83 M: 0 Drops: 0

The PDU (or Packet) List Pane at the top of the diagram displays a summary of each packet captured. By clicking on packets in this pane, you control what is displayed in the other two panes.
The PDU (or Packet) Details Pane in the middle of the diagram displays the packet selected in the Packet List Pane in more detail.
The PDU (or Packet) Bytes Pane at the bottom of the diagram displays the actual data (in hexadecimal form representing the actual binary) from the packet selected in the Packet List Pane, and highlights the field selected in the Packet Details Pane.

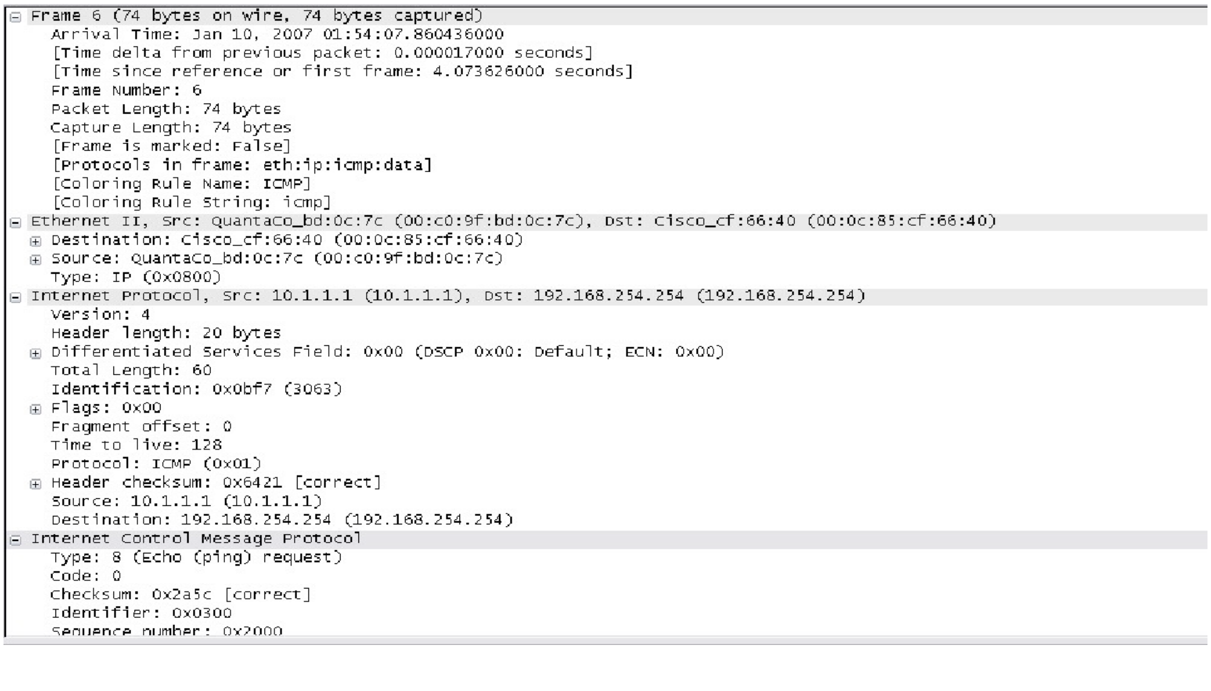
Task 1: Ping PDU Capture

Step 1: After ensuring that the standard lab topology and configuration is correct, launch Wireshark on a computer in a lab pod. Set the Capture Options as described above in the overview and start the capture process. From the command line of the computer, ping the IP address of another network connected and powered on end device on in the lab topology. In this case, ping the Eagle Server at using the command ping 192.168.254.254. After receiving the successful replies to the ping in the command line window, stop the packet capture.
Step 2: Examine the Packet List pane.
The Packet List pane on Wireshark should now look something like this:



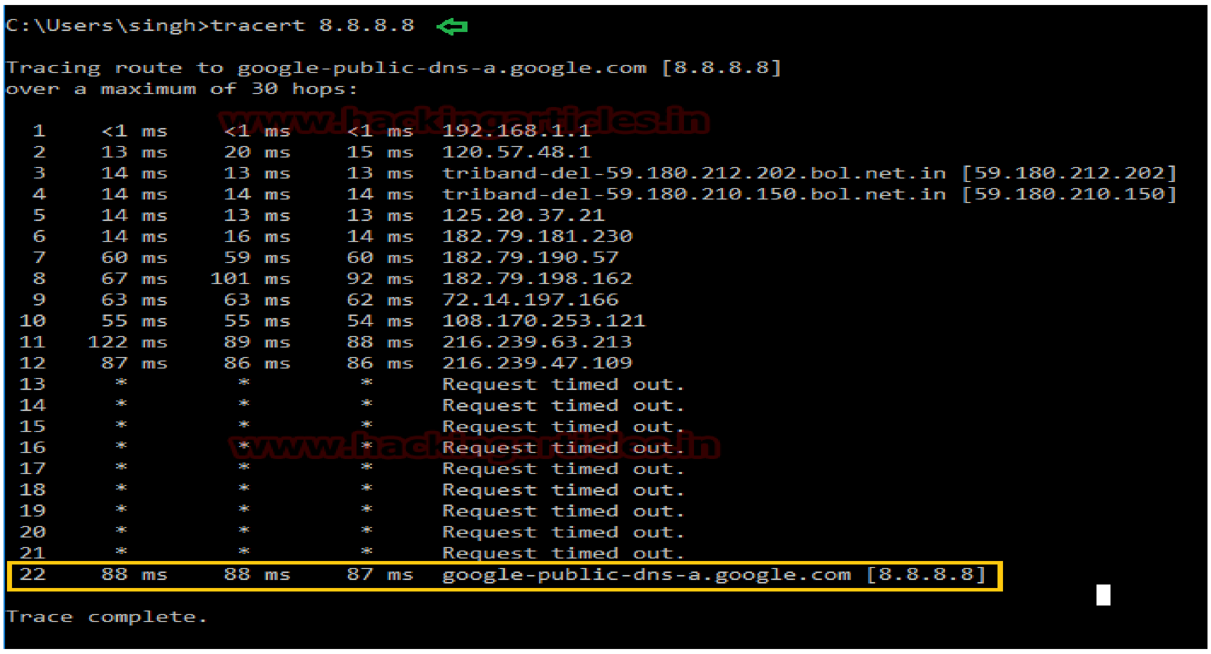
```
No. .  Time      Source            Destination         Protocol Info
     1 0.000000  Cisco_9f:6c:c9    Spanning-tree-(for  STP      Conf. Root = 32769/00:0f:f7:9f:6c:c0  Cost =
     2 2.000032  Cisco_9f:6c:c9    Spanning-tree-(for  STP      Conf. Root = 32769/00:0f:f7:9f:6c:c0  Cost =
     3 4.000059  Cisco_9f:6c:c9    Spanning-tree-(for  STP      Conf. Root = 32769/00:0f:f7:9f:6c:c0  Cost =
     4 4.072858  QuantaCo_bd:0c:7c Broadcast           ARP      Who has 10.1.1.254?  Tell 10.1.1.1
     5 4.073609  Cisco_cf:66:40    QuantaCo_bd:0c:7c   ARP      10.1.1.254 is at 00:0c:85:cf:66:40
     6 4.073626  10.1.1.1          192.168.254.254     ICMP     Echo (ping) request
     7 4.074122  192.168.254.254   10.1.1.1            ICMP     Echo (ping) reply
     8 5.067535  10.1.1.1          192.168.254.254     ICMP     Echo (ping) request
     9 5.068007  192.168.254.254   10.1.1.1            ICMP     Echo (ping) reply
    10 6.000113  Cisco_9f:6c:c9    Spanning-tree-(for  STP      Conf. Root = 32769/00:0f:f7:9f:6c:c0  Cost =
    11 6.067548  10.1.1.1          192.168.254.254     ICMP     Echo (ping) request
    12 6.068019  192.168.254.254   10.1.1.1            ICMP     Echo (ping) reply
    13 6.084103  Cisco_9f:6c:c9    Cisco_9f:6c:c9      LOOP     Reply
    14 7.067603  10.1.1.1          192.168.254.254     ICMP     Echo (ping) request
    15 7.068131  192.168.254.254   10.1.1.1            ICMP     Echo (ping) reply
    16 8.000126  Cisco_9f:6c:c9    Spanning-tree-(for  STP      Conf. Root = 32769/00:0f:f7:9f:6c:c0  Cost =
    17 9.975700  Cisco_9f:6c:c9    CDP/VTP/DTP/PAgP/U  DTP      Dynamic Trunking Protocol
    18 10.000134 Cisco_9f:6c:c9    Spanning-tree-(for  STP      Conf. Root = 32769/00:0f:f7:9f:6c:c0  Cost =
```

Look at the packets listed above; we are interested in packet numbers 6, 7, 8, 9, 11, 12, 14 and 15. Locate the equivalent packets on the packet list on your computer.
Step 3: Select (highlight) the first echo request packet on the list with the mouse. The Packet Detail pane will now display something similar to: Click on each of the four "+" to expand the information. The packet Detail Pane will now be similar to:



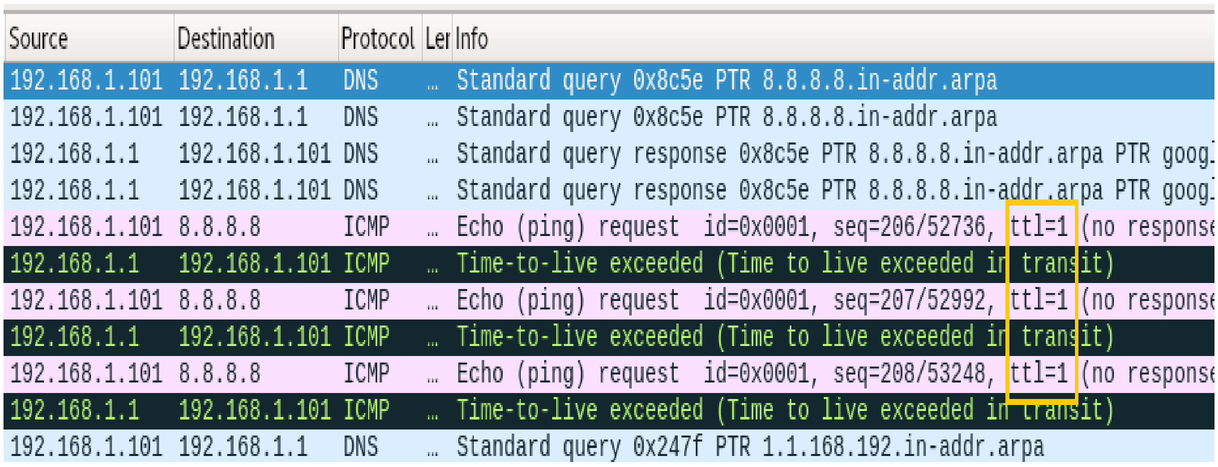
```
⊟ Frame 6 (74 bytes on wire, 74 bytes captured)
    Arrival Time: Jan 10, 2007 01:54:07.860436000
    [Time delta from previous packet: 0.000017000 seconds]
    [Time since reference or first frame: 4.073626000 seconds]
    Frame Number: 6
    Packet Length: 74 bytes
    Capture Length: 74 bytes
    [Frame is marked: False]
    [Protocols in frame: eth:ip:icmp:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp]
⊟ Ethernet II, Src: QuantaCo_bd:0c:7c (00:c0:9f:bd:0c:7c), Dst: Cisco_cf:66:40 (00:0c:85:cf:66:40)
  ⊞ Destination: Cisco_cf:66:40 (00:0c:85:cf:66:40)
  ⊞ Source: QuantaCo_bd:0c:7c (00:c0:9f:bd:0c:7c)
    Type: IP (0x0800)
⊟ Internet Protocol, Src: 10.1.1.1 (10.1.1.1), Dst: 192.168.254.254 (192.168.254.254)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 60
    Identification: 0x0bf7 (3063)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: ICMP (0x01)
  ⊞ Header checksum: 0x6421 [correct]
    Source: 10.1.1.1 (10.1.1.1)
    Destination: 192.168.254.254 (192.168.254.254)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x2a5c [correct]
    Identifier: 0x0300
    Sequence number: 0x2000
```

Task 2: Traceroute PDU Capture
Traceroute generates a list of each hop by entering IP of routers that traversed between source and destination and average round-trip time. As a result **hop 22 denotes** entry of destination i.e. Google DNS. In order to notice the activity of traceroute, Wireshark runs in the background.



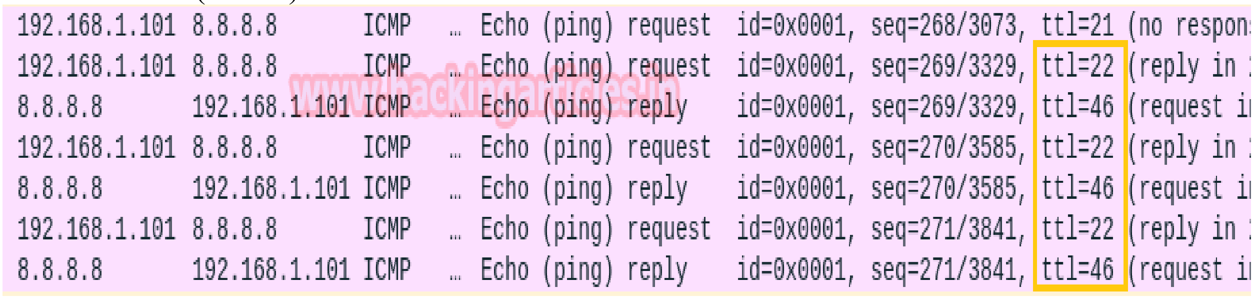
```
C:\Users\singh>tracert 8.8.8.8

Tracing route to google-public-dns-a.google.com [8.8.8.8]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  192.168.1.1
  2    13 ms    20 ms    15 ms  120.57.48.1
  3    14 ms    13 ms    13 ms  triband-del-59.180.212.202.bol.net.in [59.180.212.202]
  4    14 ms    14 ms    14 ms  triband-del-59.180.210.150.bol.net.in [59.180.210.150]
  5    14 ms    13 ms    13 ms  125.20.37.21
  6    14 ms    16 ms    14 ms  182.79.181.230
  7    60 ms    59 ms    60 ms  182.79.190.57
  8    67 ms   101 ms    92 ms  182.79.198.162
  9    63 ms    63 ms    62 ms  72.14.197.166
 10    55 ms    55 ms    54 ms  108.170.253.121
 11   122 ms    89 ms    88 ms  216.239.63.213
 12    87 ms    86 ms    86 ms  216.239.47.109
 13     *        *        *     Request timed out.
 14     *        *        *     Request timed out.
 15     *        *        *     Request timed out.
 16     *        *        *     Request timed out.
 17     *        *        *     Request timed out.
 18     *        *        *     Request timed out.
 19     *        *        *     Request timed out.
 20     *        *        *     Request timed out.
 21     *        *        *     Request timed out.
 22    88 ms    88 ms    87 ms  google-public-dns-a.google.com [8.8.8.8]

Trace complete.
```

At Wireshark the following points are notices:
- ICMP echo request packet is used instead of UDP to send DNS query.
- The packet first goes from source 192.168.1.101 to first router 192.168.1.1 having ICMP echo request packet with TTL=1
- The router will drop that packet and send ICMP Time Exceeded error message to the source.
- All this happens 3 times before the source machine sends next packet by incrementing TTL value by 1 i.e. TTL=2.



```
Source        Destination   Protocol Ler Info
192.168.1.101 192.168.1.1   DNS      …  Standard query 0x8c5e PTR 8.8.8.8.in-addr.arpa
192.168.1.101 192.168.1.1   DNS      …  Standard query 0x8c5e PTR 8.8.8.8.in-addr.arpa
192.168.1.1   192.168.1.101 DNS      …  Standard query response 0x8c5e PTR 8.8.8.8.in-addr.arpa PTR googl
192.168.1.1   192.168.1.101 DNS      …  Standard query response 0x8c5e PTR 8.8.8.8.in-addr.arpa PTR googl
192.168.1.101 8.8.8.8       ICMP     …  Echo (ping) request  id=0x0001, seq=206/52736, ttl=1 (no response
192.168.1.1   192.168.1.101 ICMP     …  Time-to-live exceeded (Time to live exceeded in transit)
192.168.1.101 8.8.8.8       ICMP     …  Echo (ping) request  id=0x0001, seq=207/52992, ttl=1 (no response
192.168.1.1   192.168.1.101 ICMP     …  Time-to-live exceeded (Time to live exceeded in transit)
192.168.1.101 8.8.8.8       ICMP     …  Echo (ping) request  id=0x0001, seq=208/53248, ttl=1 (no response
192.168.1.1   192.168.1.101 ICMP     …  Time-to-live exceeded (Time to live exceeded in transit)
192.168.1.101 192.168.1.1   DNS      …  Standard query 0x247f PTR 1.1.168.192.in-addr.arpa
```

From this image we can observe ICMP echo reply message is sent from 8.8.8.8 (destination) to 192.168.1.101 (source) for TTL 22.



```
192.168.1.101 8.8.8.8       ICMP  …  Echo (ping) request  id=0x0001, seq=268/3073, ttl=21 (no respon
192.168.1.101 8.8.8.8       ICMP  …  Echo (ping) request  id=0x0001, seq=269/3329, ttl=22 (reply in
8.8.8.8       192.168.1.101 ICMP  …  Echo (ping) reply    id=0x0001, seq=269/3329, ttl=46 (request in
192.168.1.101 8.8.8.8       ICMP  …  Echo (ping) request  id=0x0001, seq=270/3585, ttl=22 (reply in
8.8.8.8       192.168.1.101 ICMP  …  Echo (ping) reply    id=0x0001, seq=270/3585, ttl=46 (request in
192.168.1.101 8.8.8.8       ICMP  …  Echo (ping) request  id=0x0001, seq=271/3841, ttl=22 (reply in
8.8.8.8       192.168.1.101 ICMP  …  Echo (ping) reply    id=0x0001, seq=271/3841, ttl=46 (request in
```

RESULT:

Thus the capturing of ping and traceroute PDUs using a network protocol analyzer examined.

EX.NO. 6A: SIMULATING ARP PROTOCOL

AIM:

To write a socket program for simulating ARP protocol using TCP sockets in java.

ALGORITHM:

CLIENT

Step 1: Start the program
Step 2: Using socket connection is established between client and server.
Step 3: Get the IP address to be converted into MAC address.
Step 4: Send this IP address to server.
Step 5: Server returns the MAC address to client.
Step 6: Stop the program.

SERVER

Step 1: Start the program
Step 2: Accept the socket which is created by the client.
Step 3: Server maintains the table in which IP and corresponding MAC addresses are stored.
Step 4: Read the IP address which is send by the client.
Step 5: Map the IP address with its MAC address and return the MAC address to client.
Step 6: Stop the program.

Code

Server.java

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

class Server {
   public static void main(String args[]) {
      try {
         ServerSocket obj = new ServerSocket(3636);
         Socket obj1 = obj.accept();

         BufferedReader din = new BufferedReader(new
InputStreamReader(obj1.getInputStream()));
         DataOutputStream dout = new DataOutputStream(obj1.getOutputStream());

         String[] ip = {"165.165.80.80", "165.165.79.1"};
         String[] mac = {"6A:08:AA:C2", "8A:BC:E3:FA"};

         while (true) {
            String str = din.readLine();
            if (str == null) {
               break;  // Exit the loop if the input is null (e.g., client disconnects)
            }

            boolean found = false;
            for (int i = 0; i < ip.length; i++) {
               if (str.equals(ip[i])) {
                  dout.writeBytes(mac[i] + '\n');
                  found = true;
                  break;
               }
            }

            if (!found) {
               dout.writeBytes("MAC not found\n");
            }
         }

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

Client.java

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

class Client {
   public static void main(String args[]) {
      try {
         // BufferedReader for user input
         BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

         // Connect to the server on localhost at port 3636
         Socket clsct = new Socket("127.0.0.1", 3636);

         // BufferedReader to read from server input stream
         BufferedReader         din         =         new         BufferedReader(new
InputStreamReader(clsct.getInputStream()));

         // DataOutputStream to write to server output stream
         DataOutputStream dout = new DataOutputStream(clsct.getOutputStream());

         System.out.println("Enter the Logical address (IP):");
         String str1 = in.readLine();  // Read user input

         dout.writeBytes(str1 + '\n');  // Send IP to server

         // Read the response from the server
         String str = din.readLine();
         System.out.println("The Physical Address is: " + str);

         // Close the connection
         clsct.close();
```

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

EXECUTION:

Step 1: Save Server java file as Server.java and Client java file as Client.java.
Step 2: Open two cmd prompt for client and server.
Step 3: Set path for javac compiler and java interpreter in cmd prompt as follows

```
set path=" C:\Program Files\Java\jdk1.7.0_79\bin";
```

Step 4: Compile your server java program as follows

```
javac   Server.java
```

It generates Bytecode file (or Object file) as Server.class
Step 5: Compile your client java program as follows

```
javac   Client.java
```

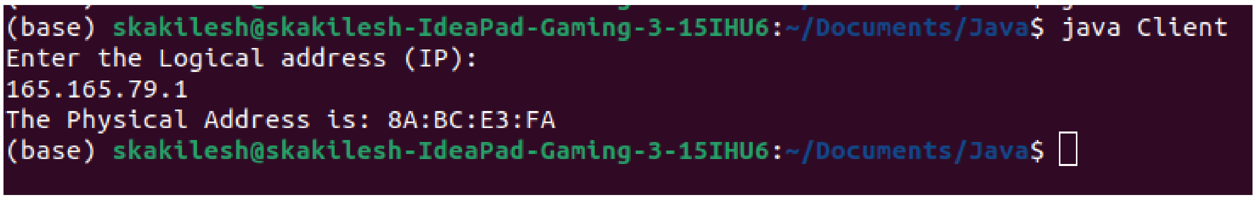
It generates Bytecode file (or Object file) as Client.class
Step 6: Execute your server java program as follows

```
java Server
```

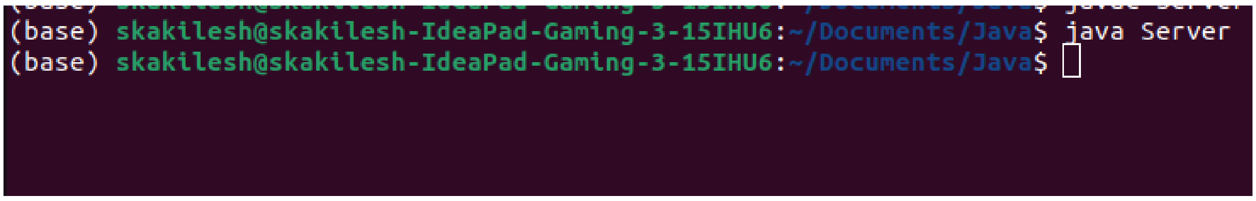
Step 7: Execute your client java program as follows

```
java Client
```

OUTPUT:



```
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java Client
Enter the Logical address (IP):
165.165.79.1
The Physical Address is: 8A:BC:E3:FA
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ 
```



```
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java Server
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ 
```

RESULT:
Thus the socket program for simulating ARP protocol using TCP sockets in java has been executed successfully and output got verified.

EX.NO. 6B: SIMULATING RARP PROTOCOL

AIM:

To write a socket program for simulating RARP protocol using TCP sockets in java.

ALGORITHM:

CLIENT

Step 1: Start the program.
Step 2: Using socket connection is established between client and server.
Step 3: Get the MAC address to be converted into IP address.
Step 4: Send this MAC address to server.
Step 5: Server returns the IP address to client.
Step 6: Stop the program.

SERVER

Step 1: Start the program
Step 2: Accept the socket which is created by the client.
Step 3: Server maintains the table in which IP and corresponding MAC addresses are stored.
Step 4: Read the MAC address which is send by the client.
Step 5: Map the IP address with its MAC address and return the IP address to client.
Step 6: Stop the program.

Code

Server.java

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

class Server {
   public static void main(String args[]) {
      try {
         // Create a server socket on port 3000
         ServerSocket obj = new ServerSocket(3000);
         System.out.println("Server started, waiting for client...");

         // Accept the incoming client connection
         Socket obj1 = obj.accept();
         System.out.println("Client connected.");

         // BufferedReader to read from client input stream
         BufferedReader din = new BufferedReader(new
InputStreamReader(obj1.getInputStream()));

         // DataOutputStream to write to client output stream
         DataOutputStream dout = new DataOutputStream(obj1.getOutputStream());

         // Define IP and MAC address pairs
         String[] ip = {"165.165.80.80", "165.165.79.1"};
         String[] mac = {"6A:08:AA:C2", "8A:BC:E3:FA"};

         while (true) {
            // Read the MAC address from the client
            String str = din.readLine();
            if (str == null) {
                break;  // Exit the loop if input is null (e.g., client disconnects)
            }

            boolean found = false;
            // Look for the corresponding IP address for the given MAC
            for (int i = 0; i < mac.length; i++) {
                if (str.equals(mac[i])) {
                    dout.writeBytes(ip[i] + '\n');
                    found = true;
                    break;
                }
            }

            if (!found) {
                dout.writeBytes("IP not found\n");
            }
        }

        // Close the connection
        obj1.close();
        obj.close();
    } catch (Exception e) {
        System.out.println(e);
    }
  }
}
```

Client.java

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

class Client {
    public static void main(String args[]) {
        try {
            // BufferedReader for user input
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

            // Create a socket to connect to the server
            Socket clsct = new Socket("127.0.0.1", 3000);

            // BufferedReader to read from the server's input stream
            BufferedReader          din          =          new          BufferedReader(new
InputStreamReader(clsct.getInputStream()));

            // DataOutputStream to write to the server's output stream
            DataOutputStream dout = new DataOutputStream(clsct.getOutputStream());

            // Prompt user to enter the MAC address
            System.out.println("Enter the Physical Address (MAC):");
            String str1 = in.readLine();  // Read user input
            // Send the MAC address to the server
            dout.writeBytes(str1 + '\n');

            // Read the server's response
            String str = din.readLine();
            System.out.println("The Logical address is (IP): " + str);

            // Close the connection
            clsct.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

EXECUTION:

Step 1: Save Server java file as Server.java and Client java file as Client.java.
Step 2: Open two cmd prompt for client and server.
Step 3: Set path for javac compiler and java interpreter in cmd prompt as follows

```
set path=" C:\Program Files\Java\jdk1.7.0_79\bin";
```

Step 4: Compile your server java program as follows

```
javac   Server.java
```

It generates Bytecode file (or Object file) as Server.class
Step 5: Compile your client java program as follows

```
javac   Client.java
```

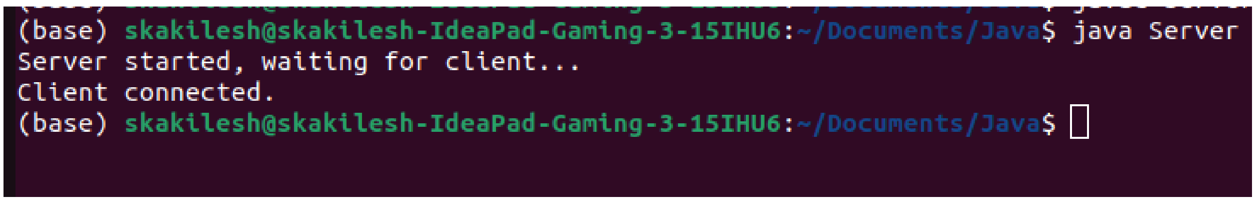
It generates Bytecode file (or Object file) as Client.class
Step 6: Execute your server java program as follows

```
java Server
```

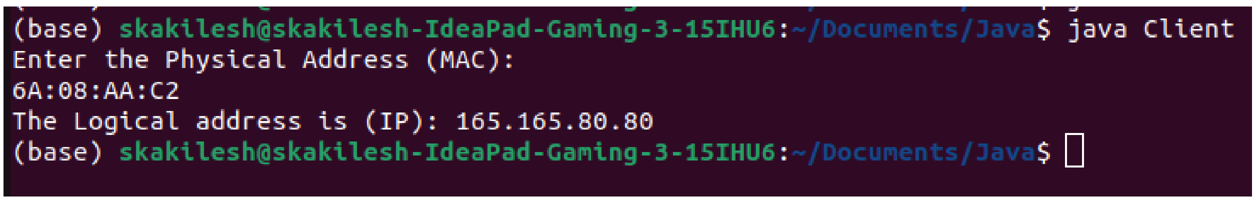
Step 7: Execute your client java program as follows

```
java Client
```

OUTPUT:



```
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java Server
Server started, waiting for client...
Client connected.
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ 
```



```
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ java Client
Enter the Physical Address (MAC):
6A:08:AA:C2
The Logical address is (IP): 165.165.80.80
(base) skakilesh@skakilesh-IdeaPad-Gaming-3-15IHU6:~/Documents/Java$ 
```

RESULT:

Thus the socket program for simulating RARP using TCP sockets in java has been executed successfully and output got verified.

EX.NO. 7. A. STUDY OF NETWORK SIMULATOR USING NS

AIM:

To study about NS2 simulator in detail.

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.

Figure 2.1 shows the basic architecture of NS2. NS2 provides users with an executable command ns which takes 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).

BASIC ARCHITECTURE:

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

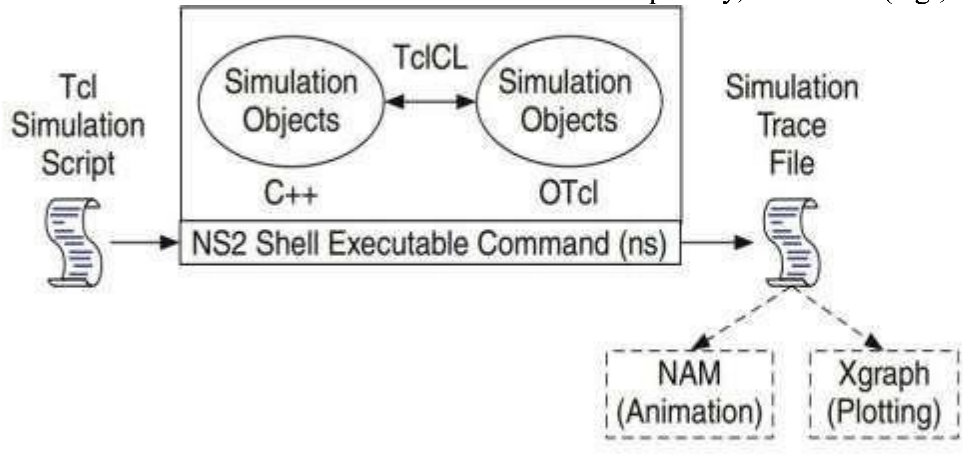


Fig. 2.1. Basic architecture of NS.

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 defines 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. We refer the readers to [14] for the detail of C++, while a brief tutorial of Tcl and OTcl tutorial are given in Appendices A.1 and A.2, respectively.
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.

1. 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.

2. Basics of TCL

Syntax: command arg1 arg2 arg3

3. 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]

4. 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

5. 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.

6. Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

7. 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, you can call this variable as you wish, 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), we need to create the files using ―open command:

#Open the Trace file
set tracefile1 [open out.tr w]
$ns trace-all $tracefile
#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

8. 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:

9. $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, we should 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. We should also define the buffer capacity of the queue related to each link. An example would be:

#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:

10. 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**

11. $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 we have several flows, we may wish to distinguish them so that we can identify 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.We shall later give the flow identification of ―2‖ to the UDP connection.

RESULT:

Thus the network simulator 2 is studied in detail.

EX.NO. 7.B. SIMULATION OF CONGESTION CONTROL ALGORITHM USING NS

AIM:

To simulate a link failure and observe the congestion control algorithm using NS2.

ALGORITHM:

Step 1: Create a simulation object
Step 2: Open the nam trace file
Step 3: Define a 'finish' procedure
Step 4: Create eight nodes
Step 5: Create links between the nodes
Step 6: Create a UDP agent and attach it to node n2,n1,n0
Step 7: Create a CBR traffic source and attach it to udp0,udp1,udp2
Step 8: Create a Null agent (a traffic sink) and attach it to node n5,n6,n7
Step 9: Connect the traffic sources with the traffic sink
Step 10: Define different colors for data flows
Step 11: Schedule events for the CBR agents
Step 12: Call the finish procedure after 5 seconds of simulation time
Step 13: Run the simulation

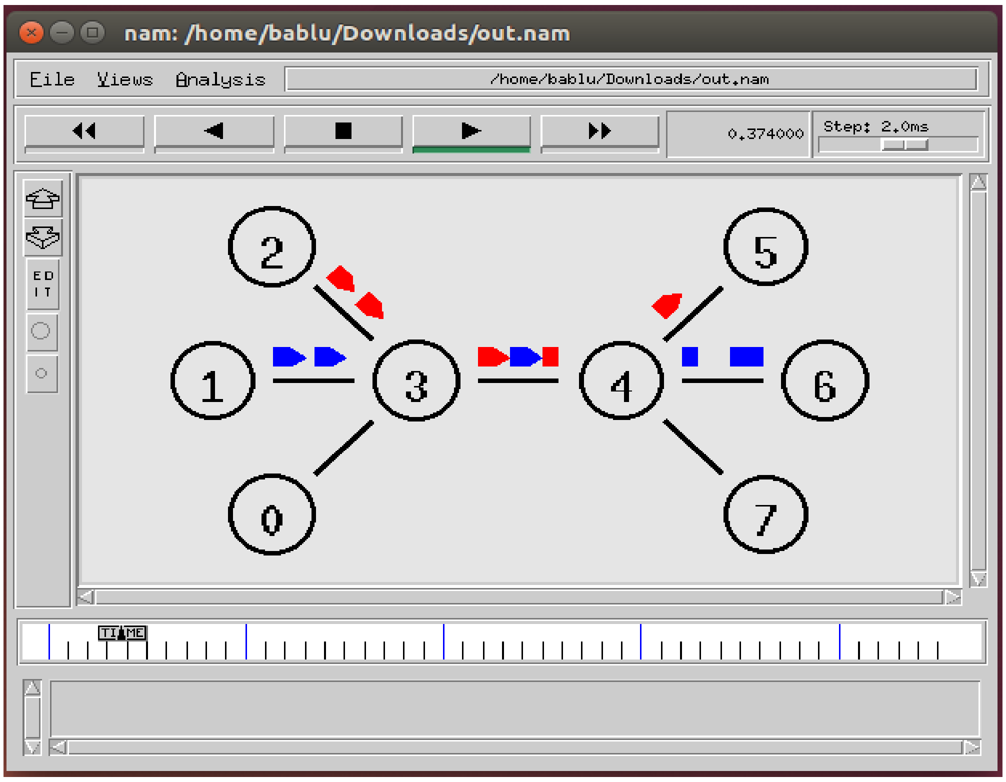
PROGRAM:

```
#Create a simulator object
set ns [new Simulator]

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

#Define a 'finish' procedure
proc finish { } {
        global ns nf
        $ns flush-trace
        #Close the trace file
        close $nf
        #Execute nam on the trace file
        exec nam out.nam &
        exit 0
}

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

#Create links between the nodes
$ns duplex-link $n0 $n3 1Mb 10ms RED
$ns duplex-link $n1 $n3 1Mb 10ms RED
$ns duplex-link $n2 $n3 1Mb 10ms RED
```

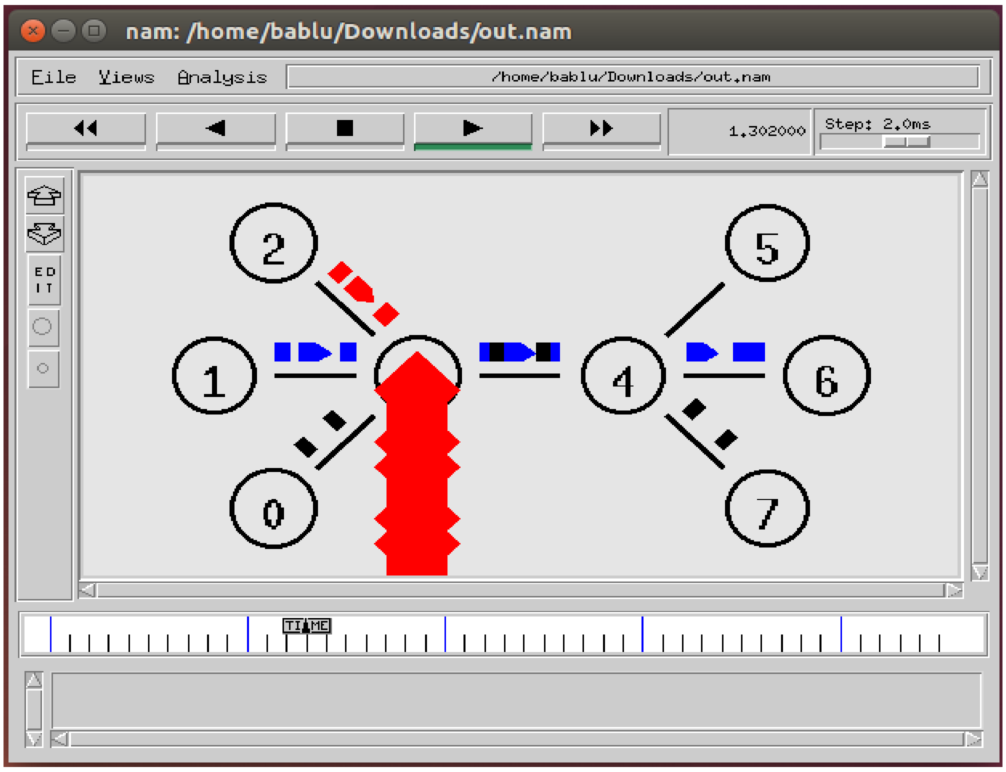
```
$ns duplex-link $n3 $n4 1Mb 10ms RED
$ns duplex-link $n4 $n5 1Mb 10ms RED
$ns duplex-link $n4 $n6 1Mb 10ms RED
$ns duplex-link $n4 $n7 1Mb 10ms RED

$ns duplex-link-op $n0 $n3 orient right-up
$ns duplex-link-op $n3 $n4 orient middle
$ns duplex-link-op $n2 $n3 orient right-down
$ns duplex-link-op $n4 $n5 orient right-up
$ns duplex-link-op $n4 $n7 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n6 $n4 orient left

#Create a UDP agent and attach it to node n2
set udp0 [new Agent/UDP]
$ns attach-agent $n2 $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 n5
set null0 [new Agent/Null]
$ns attach-agent $n5 $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0
#Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1

#Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

#Create a Null agent (a traffic sink) and attach it to node n6
set null0 [new Agent/Null]
$ns attach-agent $n6 $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp1 $null0

#Create a UDP agent and attach it to node n0
set udp2 [new Agent/UDP]
$ns attach-agent $n0 $udp2

#Create a CBR traffic source and attach it to udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packet size_ 500
$cbr2 set interval_ 0.005
$cbr2 attach-agent $udp2
```

```
#Create a Null agent (a traffic sink) and attach it to node n7
set null0 [new Agent/Null]
$ns attach-agent $n7 $null0

#Connect the traffic sources with the traffic sink
$ns connect $udp2 $null0

$udp0 set fid_ 1

$udp1 set fid_ 2

$udp2 set fid_ 3

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

#Schedule events for the CBR agents
$ns at 0.1 "$cbr0 start"
$ns at 0.2 "$cbr1 start"
$ns at 0.5 "$cbr2 start"
$ns at 4.0 "$cbr2 stop"
$ns at 4.2 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

OUTPUT:



nam: /home/bablu/Downloads/out.nam
File Views Analysis
/home/bablu/Downloads/out.nam
0.374000
Step: 2.0ms
EDIT
TIME
2
5
1
3
4
6
0
7


nam: /home/bablu/Downloads/out.nam
File Views Analysis
/home/bablu/Downloads/out.nam
1.302000
Step: 2.0ms
EDIT
TIME
2
5
1
4
6
0
7

RESULT:

Thus the congestion controlling algorithm using ns2 has been executed successfully and output got verified.

EX.NO. 8.A. STUDY OF TCP PERFORMANCE

AIM:

To study about TCP performance in detail.

Introduction :

The transmission Control Protocol (TCP) is one of the most important protocols of Internet Protocols suite. It is most widely used protocol for data transmission in communication network such as internet.

Features

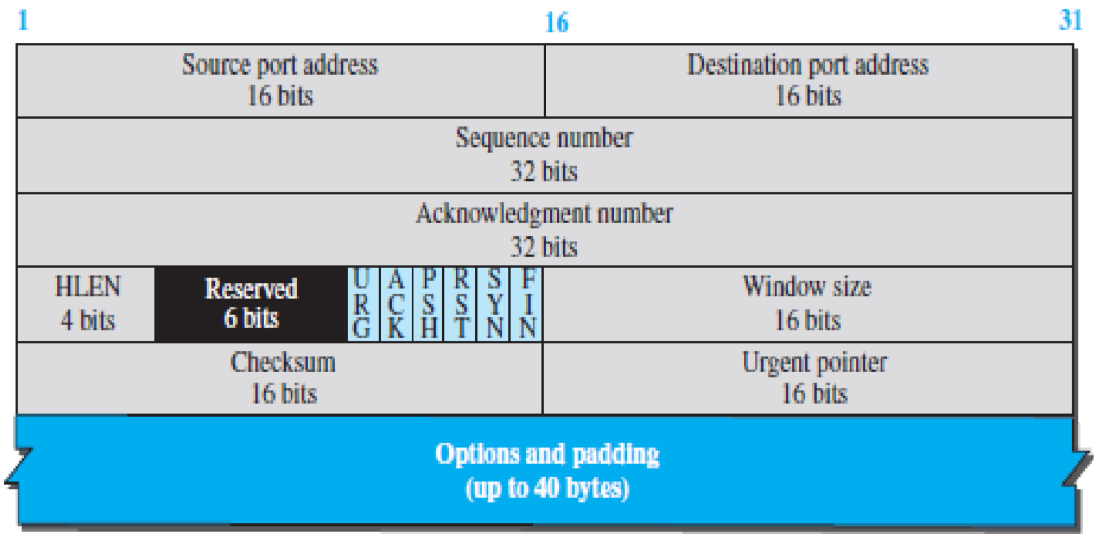
- TCP is reliable protocol. That is, the receiver always sends either positive or negative acknowledgement about the data packet to the sender, so that the sender always has bright clue about whether the data packet is reached the destination or it needs to resend it.
- TCP ensures that the data reaches intended destination in the same order it was sent.
- TCP is connection oriented. TCP requires that connection between two remote points be established before sending actual data.
- TCP provides error-checking and recovery mechanism.
- TCP provides end-to-end communication.
- TCP provides flow control and quality of service.
- TCP operates in Client/Server point-to-point mode.
- TCP provides full duplex server, i.e. it can perform roles of both receiver and sender.

Header

The length of TCP header is minimum 20 bytes long and maximum 60 bytes.

Source port address. This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.

Destination port address. This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.



Source port address. This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.

Destination port address. This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.

Sequence number. This 32-bit field defines the number assigned to the first byte of data contained in this segment. TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination which byte in this sequence is the first byte in the segment. During connection establishment each party uses a random number generator to create an **initial sequence number** (ISN), which is usually different in each direction.

Acknowledgment number. This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number *x* from the other party, it returns *x* + 1 as the acknowledgment number. Acknowledgment and data can be piggybacked together.

Header length. This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field is always between 5 (5 × 4 = 20) and 15 (15 × 4 = 60).

Control. This field defines 6 different control bits or flags. One or more of these bits can be set at a time. These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.

Window size. This field defines the window size of the sending TCP in bytes. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window (*rwnd*) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

Checksum. This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as the one described for UDP. However, the use of the checksum in the UDP datagram is optional, whereas

the use of the checksum for TCP is mandatory. The same pseudoheader, serving the same purpose, is added to the segment. For the TCP pseudoheader, the value for the protocol field is 6.

Urgent pointer. This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines a value that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment.

Options. There can be up to 40 bytes of optional information in the TCP header.

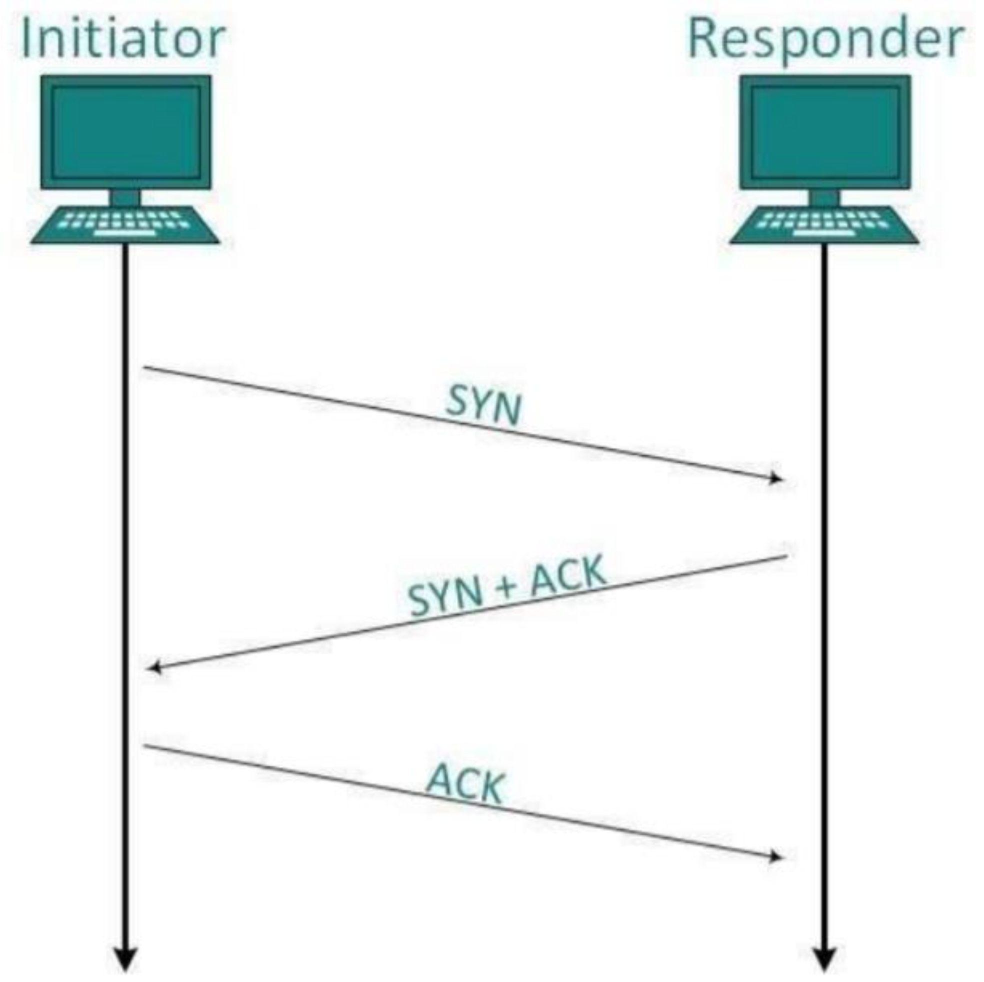
Addressing

TCP communication between two remote hosts is done by means of port numbers (TSAPs). Ports numbers can range from 0 – 65535 which are divided as:

- System Ports (0 – 1023)
- User Ports (1024 – 49151)
- Private/Dynamic Ports (49152 – 65535)

Connection Management

TCP communication works in Server/Client model. The client initiates the connection and the server either accepts or rejects it. Three-way handshaking is used for connection management.



Establishment

Client initiates the connection and sends the segment with a Sequence number. Server acknowledges it back with its own Sequence number and ACK of client's segment which is one more than client's Sequence number. Client after receiving ACK of its segment sends an acknowledgement of Server's response.

Release

Either of server and client can send TCP segment with FIN flag set to 1. When the receiving end responds it back by Acknowledging FIN, that direction of TCP communication is closed and connection is released.

Bandwidth Management

TCP uses the concept of window size to accommodate the need of Bandwidth management. Window size tells the sender at the remote end, the number of data byte segments the receiver at this end can receive. TCP uses slow start phase by using window size 1 and increases the window size exponentially after each successful communication.

For example, the client uses windows size 2 and sends 2 bytes of data. When the acknowledgement of this segment received the windows size is doubled to 4 and next sent the segment sent will be 4 data bytes long. When the acknowledgement of 4-byte data segment is received, the client sets windows size to 8 and so on.

If an acknowledgement is missed, i.e. data lost in transit network or it received NACK, then the window size is reduced to half and slow start phase starts again.

Error Control &and Flow Control

TCP uses port numbers to know what application process it needs to handover the data segment. Along with that, it uses sequence numbers to synchronize itself with the remote host. All data segments are sent and received with sequence numbers. The Sender knows which last data segment was received by the Receiver when it gets ACK. The Receiver knows about the last segment sent by the Sender by referring to the sequence number of recently received packet.

If the sequence number of a segment recently received does not match with the sequence number the receiver was expecting, then it is discarded and NACK is sent back. If two segments arrive with the same sequence number, the TCP timestamp value is compared to make a decision.

Multiplexing

The technique to combine two or more data streams in one session is called Multiplexing. When a TCP client initializes a connection with Server, it always refers to a welldefined port number which indicates the application process. The client itself uses a randomly generated port number from private port number pools.

Using TCP Multiplexing, a client can communicate with a number of different application process in a single session. For example, a client requests a web page which in turn contains different types of data (HTTP, SMTP, FTP etc.) the TCP session timeout is increased and the session is kept open for longer time so that the three-way handshake overhead can be avoided.

This enables the client system to receive multiple connection over single virtual connection. These virtual connections are not good for Servers if the timeout is too long.

Congestion Control

When large amount of data is fed to system which is not capable of handling it, congestion occurs. TCP controls congestion by means of Window mechanism. TCP sets a window size telling the other end how much data segment to send. TCP may use three algorithms for congestion control:

· Additive increase, Multiplicative Decrease
· Slow Start
· Timeout React

Timer Management

TCP uses different types of timer to control and management various tasks:

Keep-alive timer:

· This timer is used to check the integrity and validity of a connection.
· When keep-alive time expires, the host sends a probe to check if the connection still exists.

Retransmission timer:

· This timer maintains stateful session of data sent.
· If the acknowledgement of sent data does not receive within the Retransmission time, the data segment is sent again.

Persist timer:

· TCP session can be paused by either host by sending Window Size 0.
· To resume the session a host needs to send Window Size with some larger value. · If this segment never reaches the other end, both ends may wait for each other for infinite time.
· When the Persist timer expires, the host re-sends its window size to let the other end know.
· Persist Timer helps avoid deadlocks in communication.

Timed-Wait:

· After releasing a connection, either of the hosts waits for a Timed-Wait time to terminate the connection completely.
· This is in order to make sure that the other end has received the acknowledgement of its connection termination request.
· Timed-out can be a maximum of 240 seconds (4 minutes).

Crash Recovery

TCP is very reliable protocol. It provides sequence number to each of byte sent in segment. It provides the feedback mechanism i.e. when a host receives a packet, it is bound to ACK that packet having the next sequence number expected (if it is not the last segment).

When a TCP Server crashes mid-way communication and re-starts its process it sends TPDU broadcast to all its hosts. The hosts can then send the last data segment which was never unacknowledged and carry onwards.

Algorithm

1. Create a simulator object
2. Define different flows for data flows
3. Trace all events in a nam file and text file
4. Create source nodes (s1, s2, s3), gateway (G) and receiver (r)
5. Describe their layout topology
6. Specify the link between nodes
7. Define the queue size between nodes G and r as 5
8. Monitor queue on all links vertically 90°
9. Create TCP agents tcp1, tcp2, tcp3 and attach it to nodes s1, s2 and s3 respectively
10. Create three TCP sinks and attach it to node r
11. Connect traffic sources to the sink
12. Create FTP agents ftp1, ftp2, ftp3 and attach it to tcp1, tcp2 and tcp3 respectively
13. Label the nodes at start time
14. Schedule ftp1, ftp2, ftp3 to start at 0.1 and stop at 5.0 seconds
15. Call finish procedure at 5.25 seconds
16. Run the simulation
17. Execute NAM on the trace file
18. Observe the simulated events on the NAM editor and packet flow on link G to r
19. View the trace file and analyse the events
20. Stop

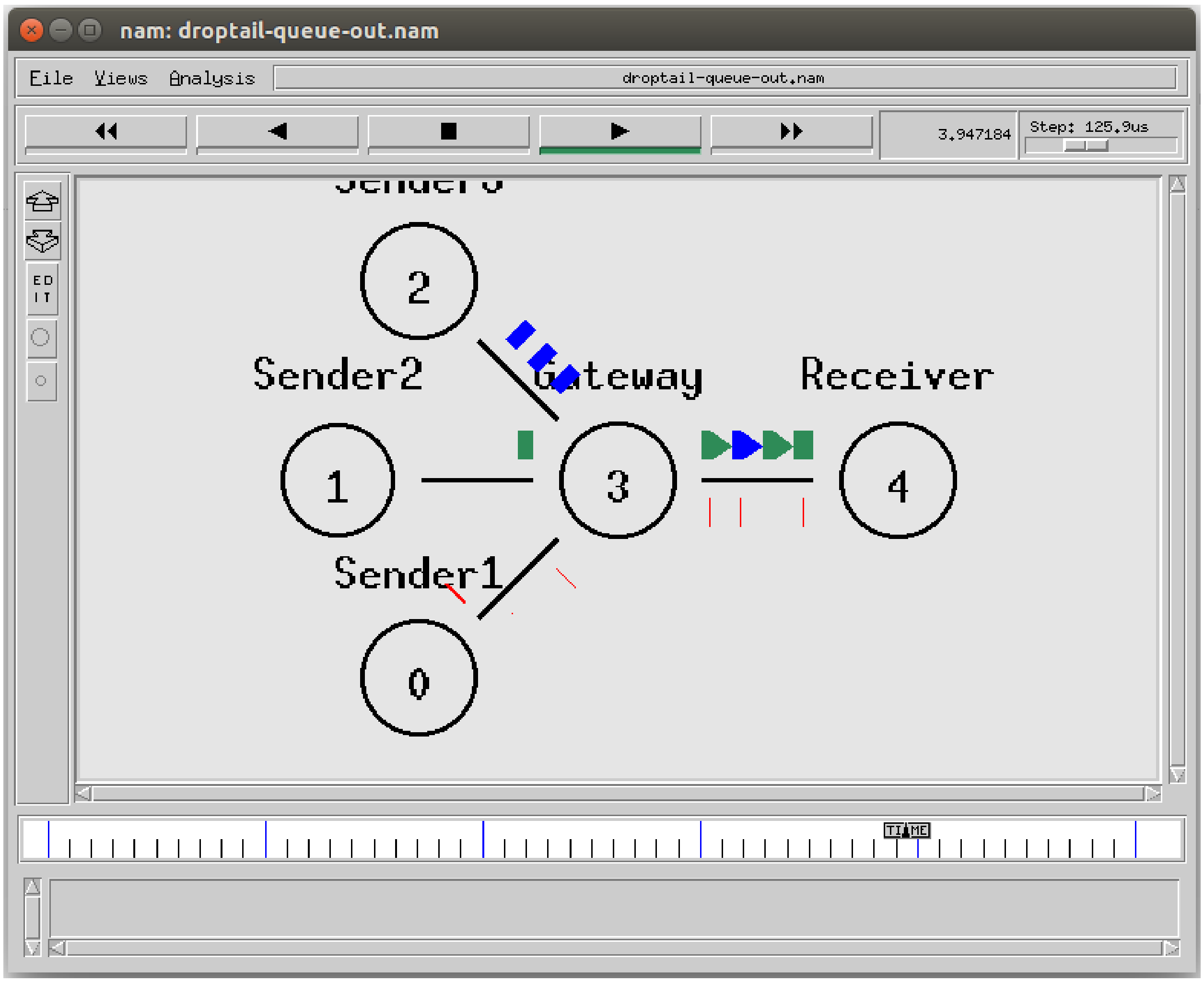
PROGRAM : File name - TCP.tcl

```
#Create a simulator object
set ns [new Simulator]

#Open trace files
set f [open droptail-queue-out.tr w]
$ns trace-all $f

#Open the nam trace file
set nf [open droptail-queue-out.nam w]
$ns namtrace-all $nf

#s1, s2 and s3 act as sources.
set s1 [$ns node]
set s2 [$ns node]
set s3 [$ns node]

#G acts as a gateway
set G [$ns node]

#r acts as a receiver
set r [$ns node]

#Define different colors for data flows
$ns color 1 red
```

```
$ns color 2 SeaGreen
$ns color 3 blue

#Create links between the nodes
$ns duplex-link $s1 $G 6Mb 10ms DropTail
$ns duplex-link $s2 $G 6Mb 10ms DropTail
$ns duplex-link $s3 $G 6Mb 10ms DropTail
$ns duplex-link $G $r 3Mb 10ms DropTail

#Define the layout of the nodes
$ns duplex-link-op $s1 $G orient right-up
$ns duplex-link-op $s2 $G orient right
$ns duplex-link-op $s3 $G orient right-down
$ns duplex-link-op $G $r orient right

#Define the queue size for the link between node G and r
$ns queue-limit $G $r 5

#Monitor the queues for links vertically
$ns duplex-link-op $s1 $G queuePos 0.5
$ns duplex-link-op $s2 $G queuePos 0.5
$ns duplex-link-op $s3 $G queuePos 0.5
$ns duplex-link-op $G $r queuePos 0.5

#Create a TCP agent and attach it to node s1
set tcp1 [new Agent/TCP/Reno]
$ns attach-agent $s1 $tcp1
$tcp1 set window_ 8
$tcp1 set fid_ 1

#Create a TCP agent and attach it to node s2
set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $s2 $tcp2
$tcp2 set window_ 8
$tcp2 set fid_ 2

#Create a TCP agent and attach it to node s3
set tcp3 [new Agent/TCP/Reno]
$ns attach-agent $s3 $tcp3
$tcp3 set window_ 4
$tcp3 set fid_ 3

#Create TCP sink agents and attach them to node r
set sink1 [new Agent/TCPSink]
set sink2 [new Agent/TCPSink]
set sink3 [new Agent/TCPSink]
$ns attach-agent $r $sink1
$ns attach-agent $r $sink2
$ns attach-agent $r $sink3

#Connect the traffic sources with the traffic sinks
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2
$ns connect $tcp3 $sink3
```

```
#Create FTP applications and attach them to agents
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3

#Define a 'finish' procedure
proc finish {} {
 global ns
 $ns flush-trace
 puts "running nam..."
 exec nam -a droptail-queue-out.nam &
 exit 0
}

#Define label for nodes
$ns at 0.0 "$s1 label Sender1"
$ns at 0.0 "$s2 label Sender2"
$ns at 0.0 "$s3 label Sender3"
$ns at 0.0 "$G label Gateway"
$ns at 0.0 "$r label Receiver"

#Schedule ftp events
$ns at 0.1 "$ftp1 start"
$ns at 0.1 "$ftp2 start"
$ns at 0.1 "$ftp3 start"
$ns at 5.0 "$ftp1 stop"
$ns at 5.0 "$ftp2 stop"
$ns at 5.0 "$ftp3 stop"

#Call finish procedure after 5 seconds of simulation time
$ns at 5.25 "finish"

#Run the simulation
$ns run
```

OUTPUT:



RESULT:

Thus the TCP performance is studied in detail.

EX.NO.8.B. STUDY OF UDP PERFORMANCE

AIM:

To study about UDP performance in detail.

Introduction :

The User Datagram Protocol (UDP) is simplest Transport Layer communication protocol available of the TCP/IP protocol suite. It involves minimum amount of communication mechanism. UDP is said to be an unreliable transport protocol but it uses IP services which provides best effort delivery mechanism.

In UDP, the receiver does not generate an acknowledgement of packet received and in turn, the sender does not wait for any acknowledgement of packet sent. This shortcoming makes this protocol unreliable as well as easier on processing.

Requirement of UDP

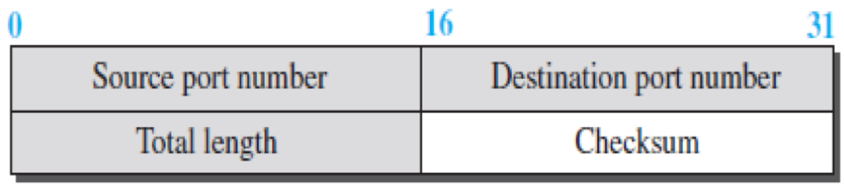
A question may arise, why do we need an unreliable protocol to transport the data? We deploy UDP where the acknowledgement packets share significant amount of bandwidth along with the actual data. For example, in case of video streaming, thousands of packets are forwarded towards its users. Acknowledging all the packets is troublesome and may contain huge amount of bandwidth wastage. The best delivery mechanism of underlying IP protocol ensures best efforts to deliver its packets, but even if some packets in video streaming get lost, the impact is not calamitous and can be ignored easily. Loss of few packets in video and voice traffic sometimes goes unnoticed.

Features

· UDP is used when acknowledgement of data does not hold any significance.
· UDP is good protocol for data flowing in one direction.
· UDP is simple and suitable for query based communications.
· UDP is not connection oriented.
· UDP does not provide congestion control mechanism.
· UDP does not guarantee ordered delivery of data.
· UDP is stateless.
· UDP is suitable protocol for streaming applications such as VoIP, multimedia streaming.

UDP Header

UDP header is as simple as its function.



UDP header contains four main parameters:
· **Source Port** - This 16 bits information is used to identify the source port of the packet.
· **Destination Port** - This 16 bits information, is used identify application level service on destination machine.
· **Length -** Length field specifies the entire length of UDP packet (including header). It is 16-bits field and minimum value is 8-byte, i.e. the size of UDP header itself.
· **Checksum** - This field stores the checksum value generated by the sender before sending. IPv4 has this field as optional so when checksum field does not contain any value it is made 0 and all its bits are set to zero.

UDP application

Here are few applications where UDP is used to transmit data:

· Domain Name Services
· Simple Network Management Protocol
· Trivial File Transfer Protocol
· Routing Information Protocol
· Kerberos

Algorithm

1. Create a simulator object
2. Define different color for data flows
3. Trace all events in a nam file.
4. Create four nodes n0, n1, n2 and n3
5. Describe their layout topology
6. Specify the link capacity between nodes
7. Monitor queue on the link n2 to n3 vertically 90°
8. Create a UDP agents udp0, udp1 and attach it to nodes n0 and n1 respectively
9. Create a CBR traffic cbr0, cbr1 and attach it to udp0 and udp1 respectively
10. Create a traffic sink and attach it to node n3
11. Connect sources to the sink
12. Label the nodes
13. Schedule cbr0 to start at 0.5 and stop at 4.5 seconds
14. Schedule cbr1 to start at 1.0 and stop at 4.0 seconds
15. Call finish procedure at 5.0 seconds
16. Run the simulation
17. Execute NAM on the trace file
18. Observe simulated events on the NAM and packet flow on link n2 to n3
19. Stop

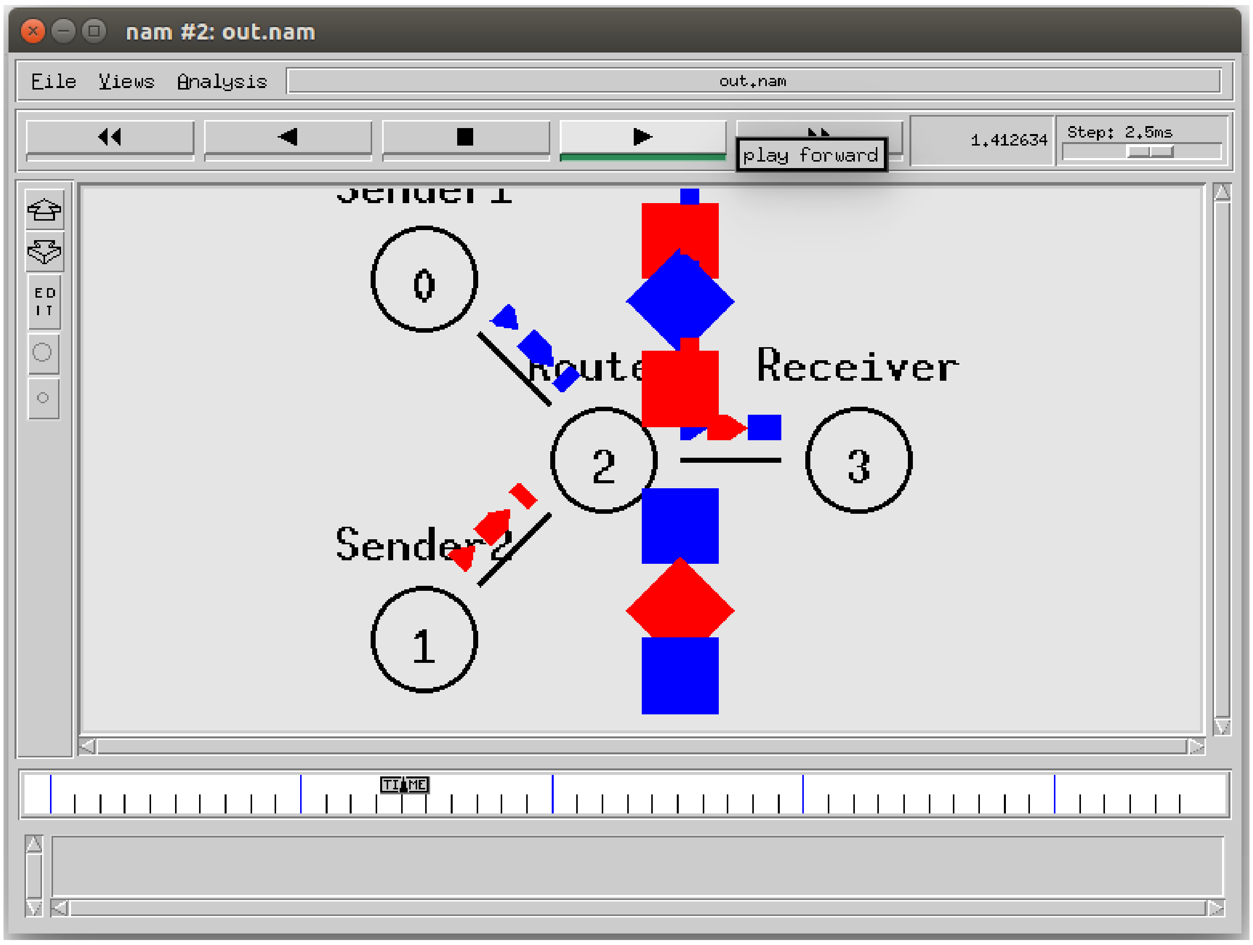
PROGRAM : File name - UDP.tcl

```
#Create a simulator object
set ns [new Simulator]

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

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

#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms SFQ

#Specify layout of nodes
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
```

```
$ns duplex-link-op $n2 $n3 orient right

#Monitor the queue for the link 2ņ3 vertically
$ns duplex-link-op $n2 $n3 queuePos 0.5

#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$udp0 set class_ 1
$ns attach-agent $n0 $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 UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$udp1 set class_ 2
$ns attach-agent $n1 $udp1

# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

#Create a Null agent (a traffic sink) and attach it to node n3
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0

#Connect traffic sources with the traffic sink
$ns connect $udp0 $null0
$ns connect $udp1 $null0

#Define finish procedure
proc finish {} {
 global ns nf
 $ns flush-trace
 #Close the trace file
 close $nf
 #Execute nam on the trace file
 exec nam -a out.nam &
 exit 0
}

#Define label for nodes
$ns at 0.0 "$n0 label Sender1"
$ns at 0.0 "$n1 label Sender2"
$ns at 0.0 "$n2 label Router"
$ns at 0.0 "$n3 label Receiver"

#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
```

```
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"

#Call finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

OUTPUT:



RESULT:

Thus the UDP performance is studied in detail.

EX.NO. 9.A. SIMULATION OF DISTANCE VECTOR ROUTING ALGORITHM

AIM:

To simulate and observe traffic route of a network using distance vector routing protocol.

ALGORITHM:

Step 1: Create a simulator object
Step 2: Set routing protocol to Distance vector routing
Step 3: Trace packets on all links on to NAM trace and text trace file.
Step 4: Define finish procedure to close files, flash tracing and run NAM
Step 5: Create 5 nodes
Step 6: Specify the link characteristics between the nodes
Step 7: Describer their layout topology as a octagon
Step 8: Add UDP agent for node n0
Step 9: Create CBR traffic on the top of UDP and set traffic parameters
Step 10: Add NULL agent to node n3
Step 11: Connect source and sink
Step 12: Schedule as follows

- Start traffic flow at 1.0
- Down the link n1 – n2 at 15.0
- Up the link n1 – n2 at 25.0
- Call finish procedure at 35.0

Step 13: Start the scheduler
Step 14: Observe the traffic route when the link is up and down
Step 15: View the simulated events and trace file analyze it
Step 16: Stop.

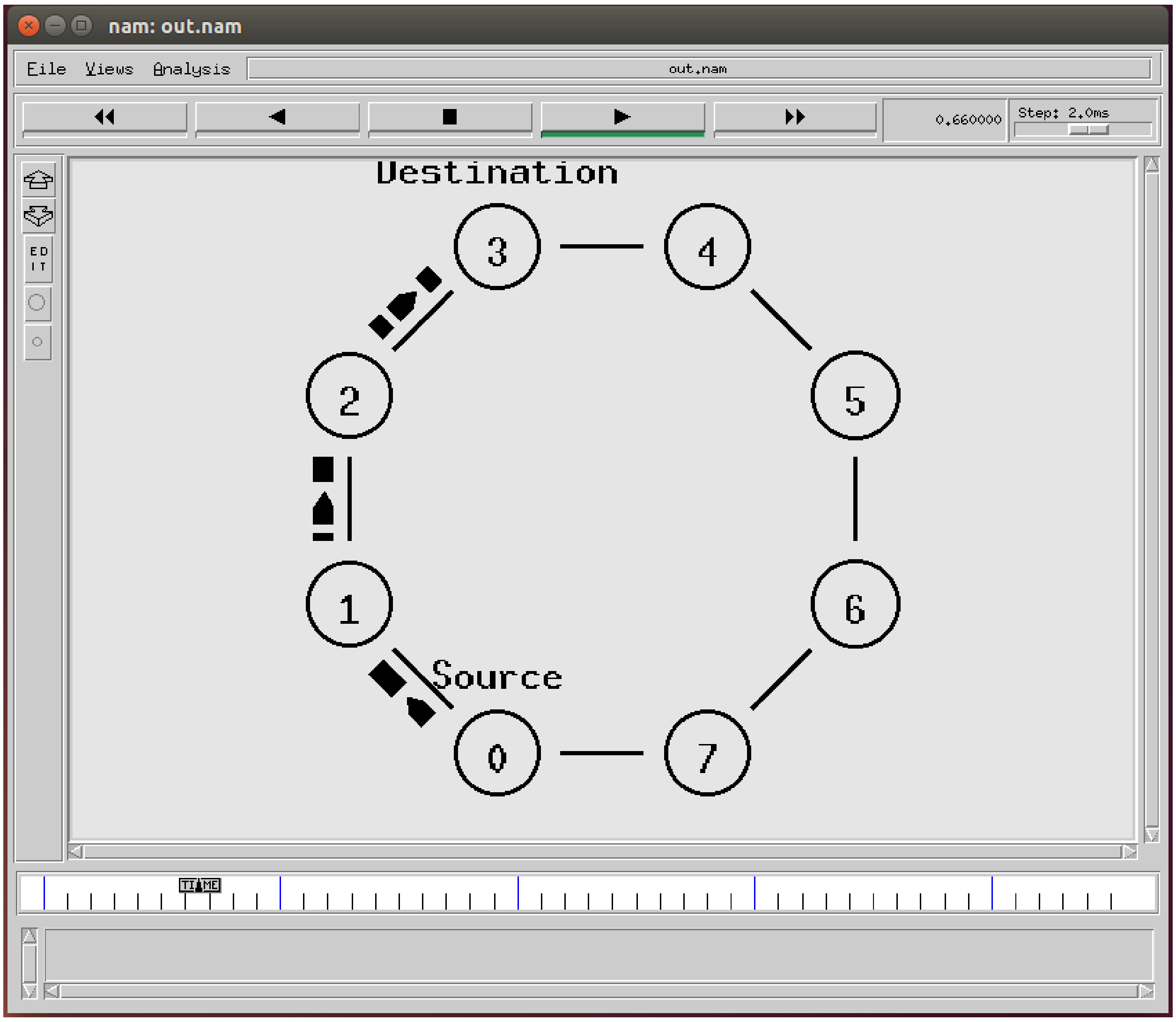
PROGRAM:

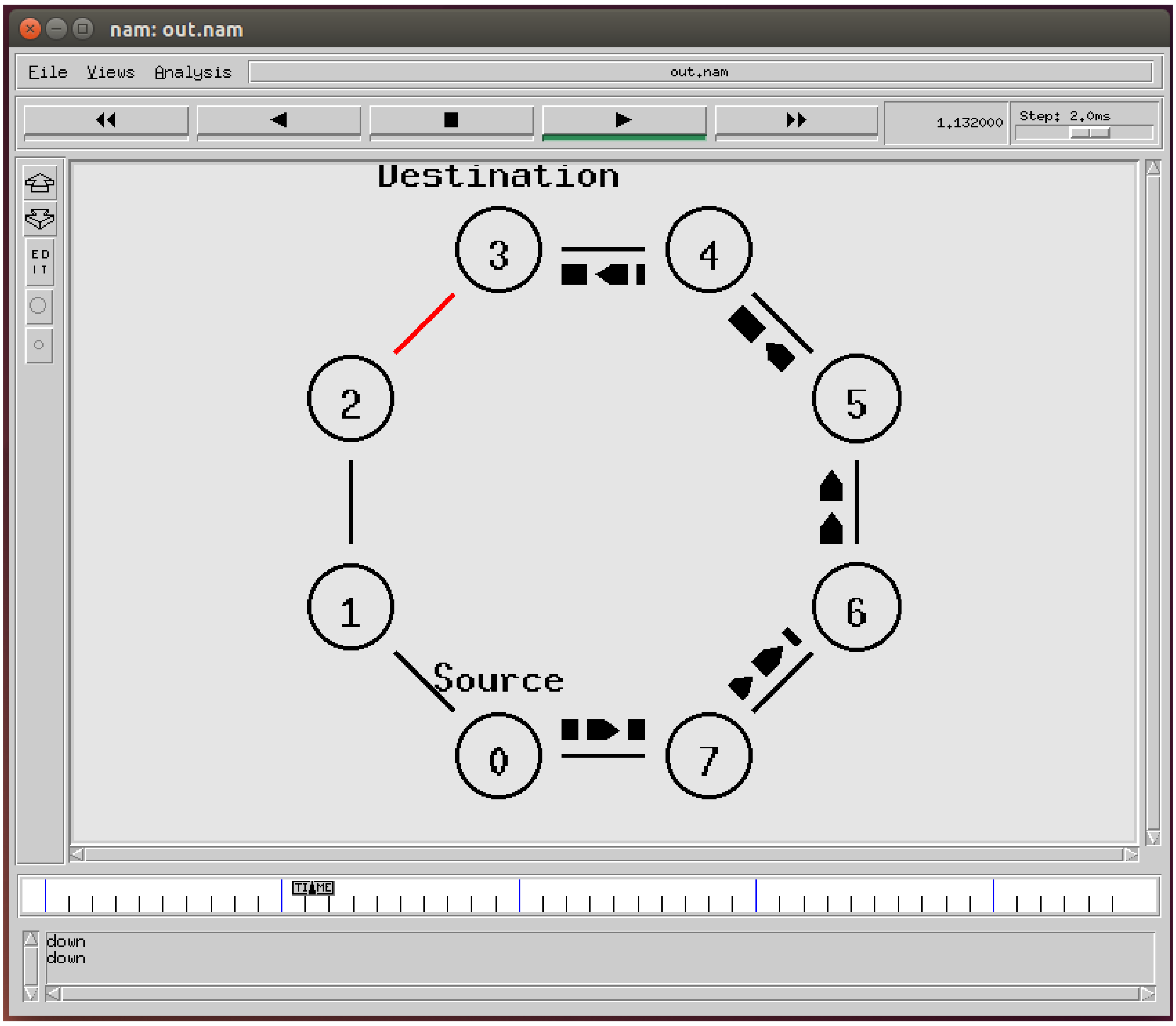
```
#Create a simulator object
set ns [new Simulator]

#Use distance vector routing
$ns rtproto DV

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

# Open tracefile
set nt [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
        #Execute nam on the trace file
        exec nam -a out.nam &
        exit 0
}

# Create 8 nodes
```

```
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]

# 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:





down
down

RESULT:

Thus the simulation of distance vector routing algorithm using ns2 has been executed successfully and output got verified.

EX.NO. 9.B. SIMULATION OF LINK STATE ROUTING ALGORITHM

AIM:

To simulate and observe traffic route of a network using link state routing algorithm.

ALGORITHM:

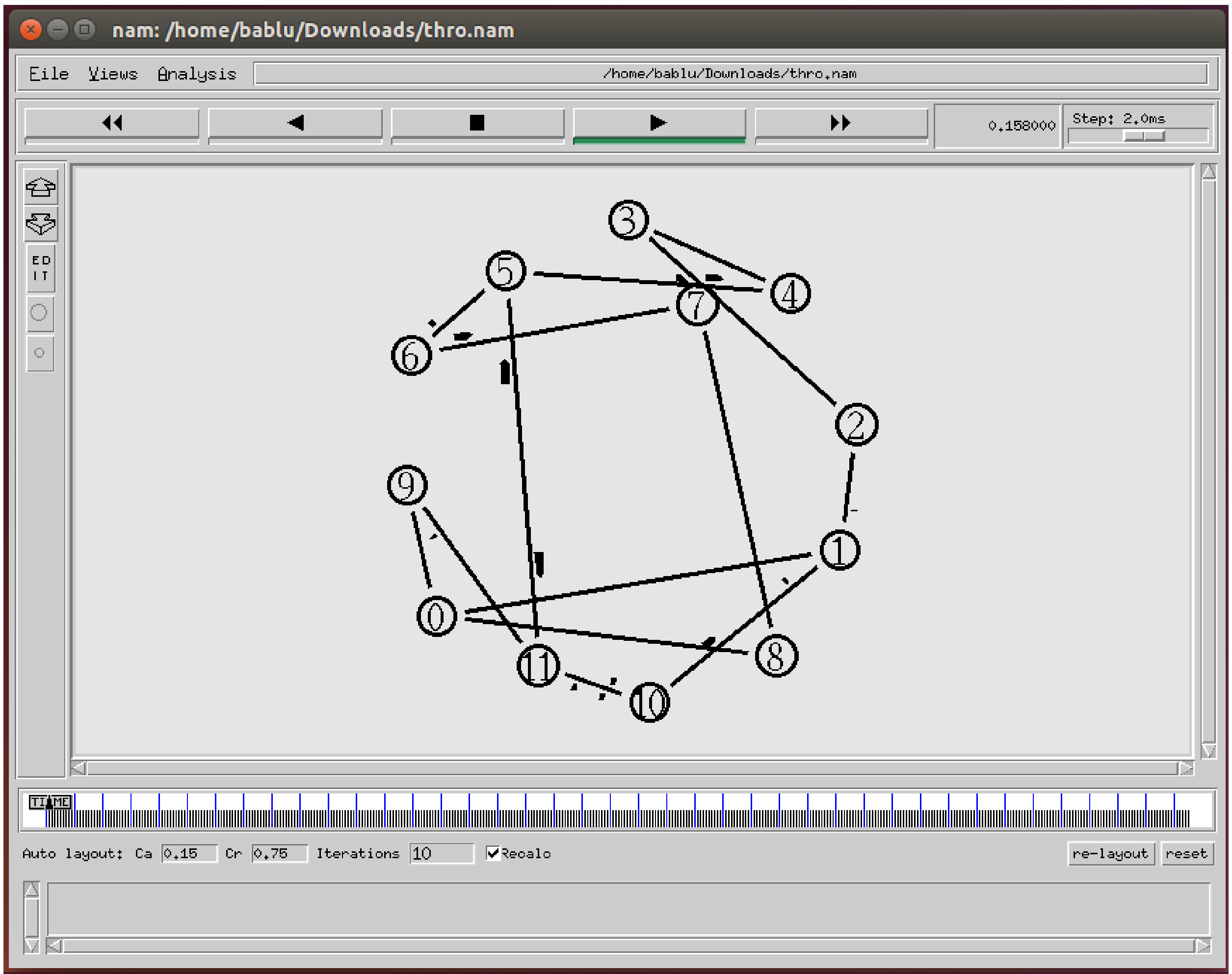
Step 1: Create a simulator object
Step 2: Trace packets on all links on to NAM trace and text trace file.
Step 3: Define finish procedure to close files, flash tracing and run NAM
Step 4: Create 5 nodes
Step 5: Specify the link characteristics between the nodes
Step 6: Describer their layout topology as a octagon
Step 7: Add UDP agent for node n0
Step 8: Create CBR traffic on the top of UDP and set traffic parameters
Step 9: Add NULL agent to node n3
Step 10: Connect source and sink
Step 11: Set routing protocol to Distance vector routing
Step 12: Start the scheduler
Step 13: Observe the traffic route when the link is up and down
Step 14: View the simulated events and trace file analyze it
Step 15: Stop.

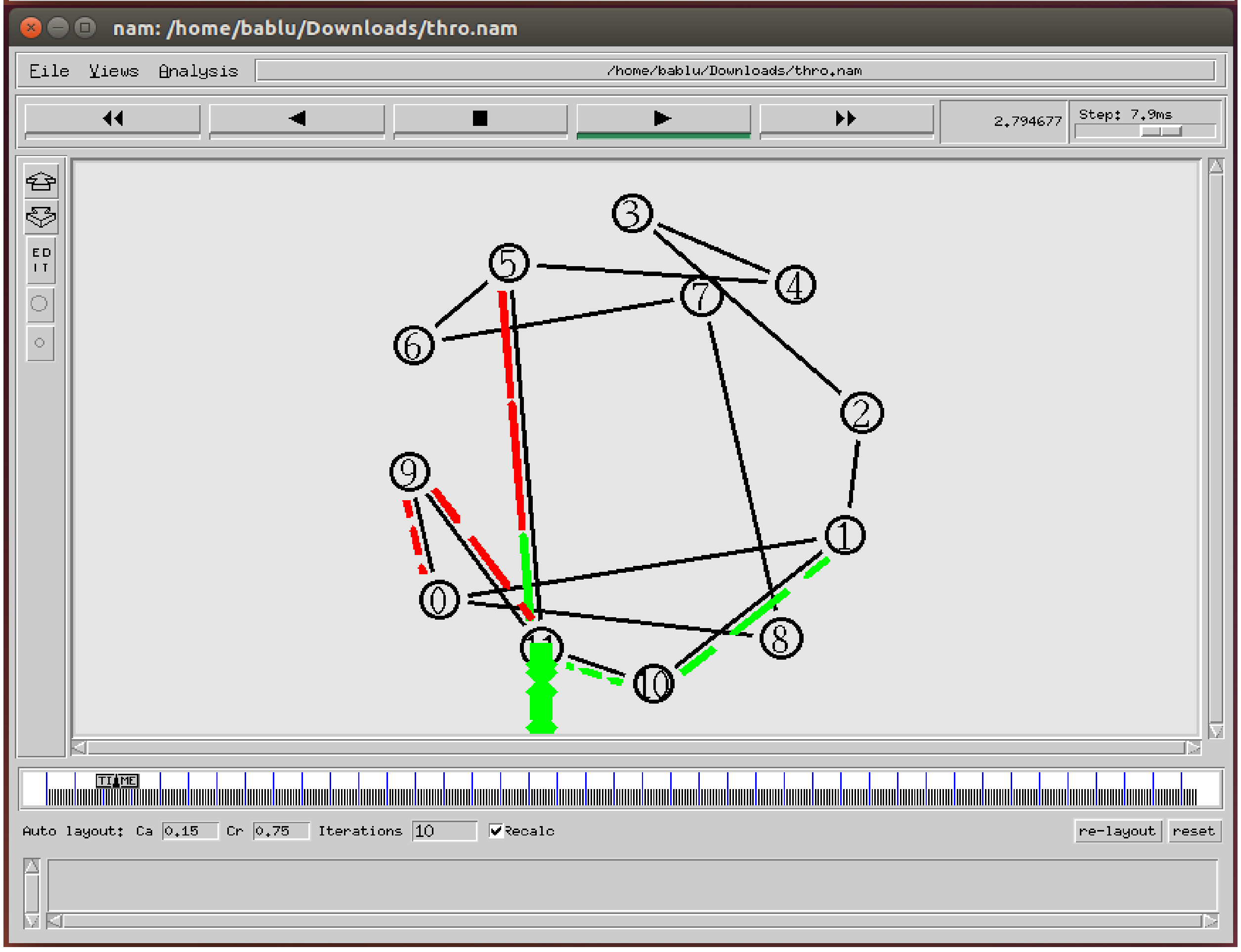
PROGRAM:

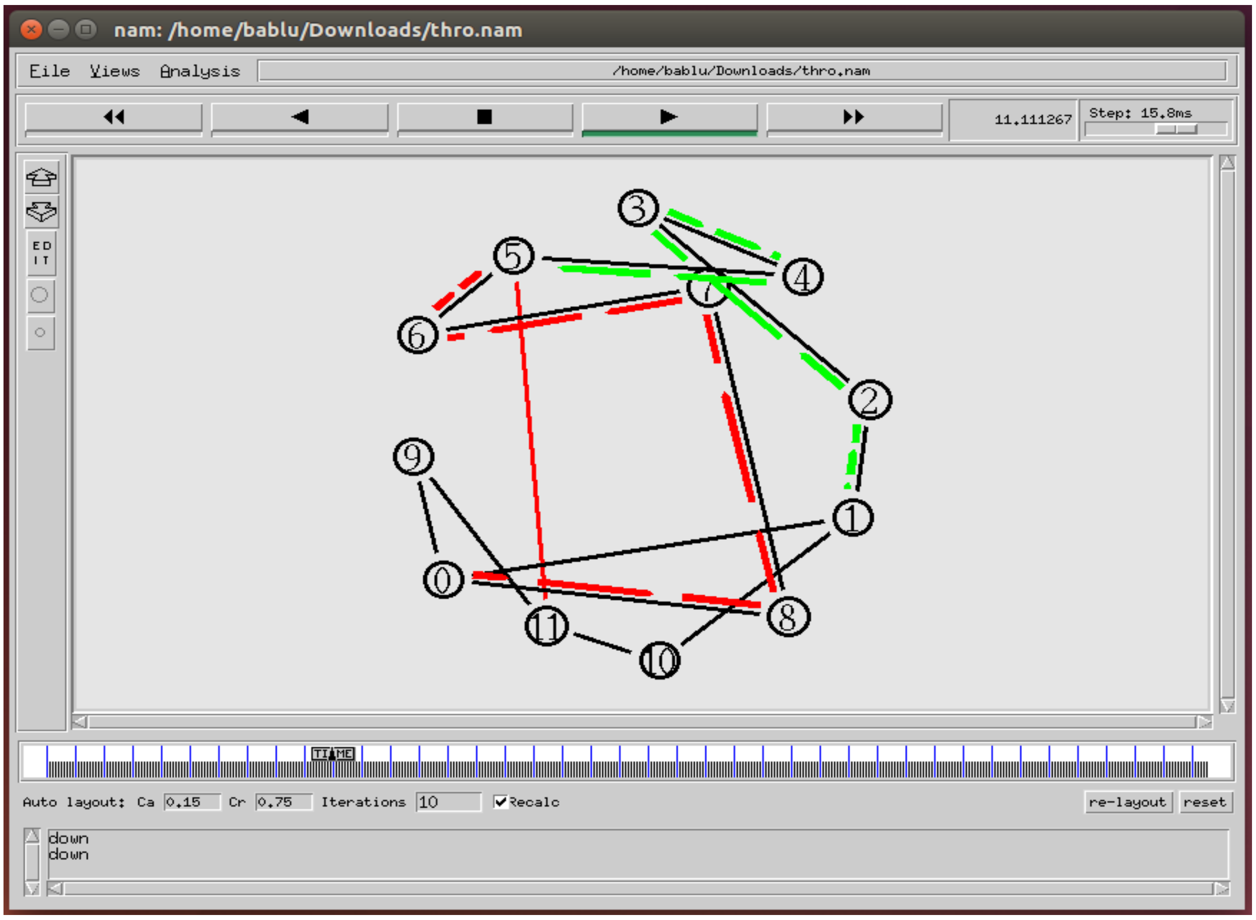
```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]

$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 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:


nam: /home/bablu/Downloads/thro.nam
File
Views
Analysis
/home/bablu/Downloads/thro.nam
0.158000
Step: 2.0ms
EDIT
TIME
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset
3
5
7
4
6
2
9
1
0
11
8
10


nam: /home/bablu/Downloads/thro.nam
File
Views
Analysis
/home/bablu/Downloads/thro.nam
2.794677
Step: 7.9ms
EDIT
TIME
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset
3
5
7
4
6
2
9
1
0
11
8
10



Auto layout: Ca 0.15 Cr 0.75 Iterations 10 ✓Recalc

```
down
down
```

RESULT:

Thus the simulation of link state routing algorithm using ns2 has been executed successfully and output got verified.

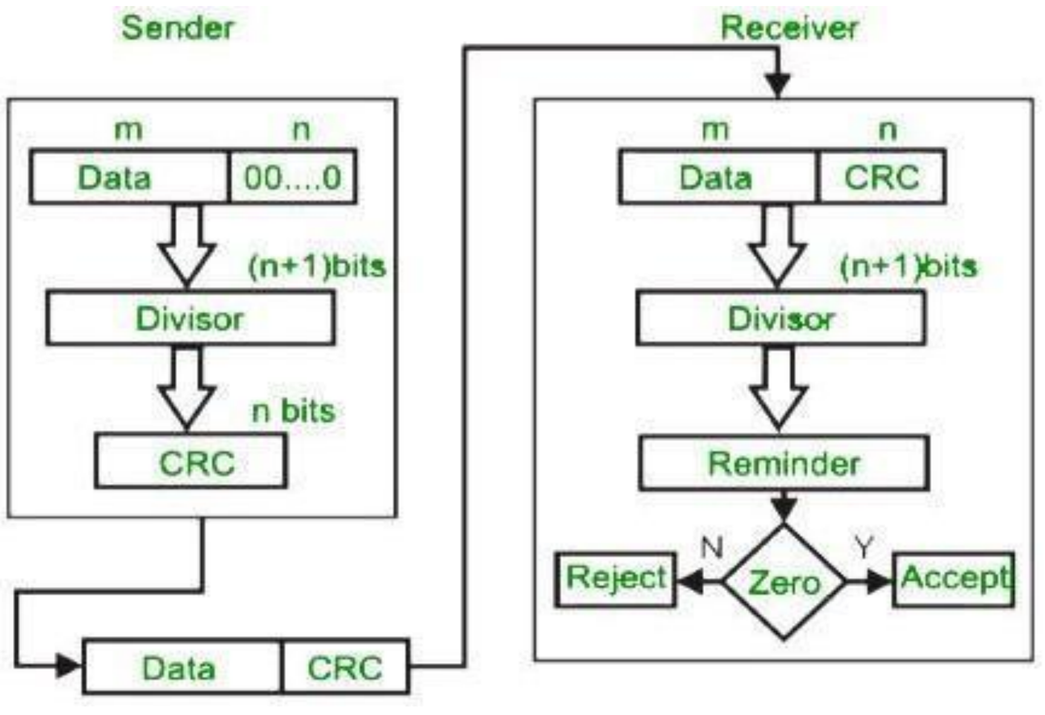
EX.NO. 10. SIMULATION OF ERROR CORRECTION CODE (CRC)

AIM:

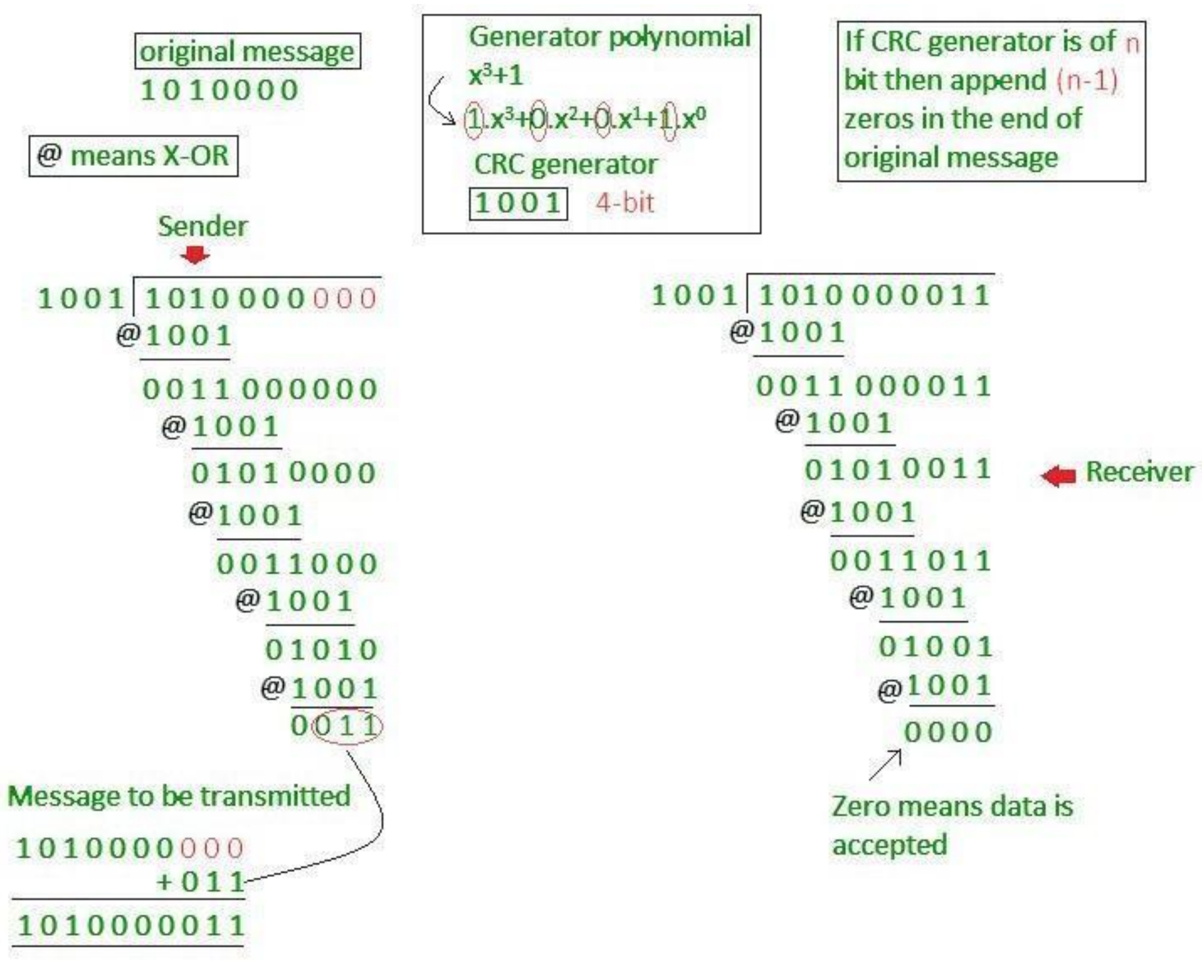
To simulate the error correction code like CRC using java.

Cyclic redundancy check (CRC)

- Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
- A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.



EXAMPLE :



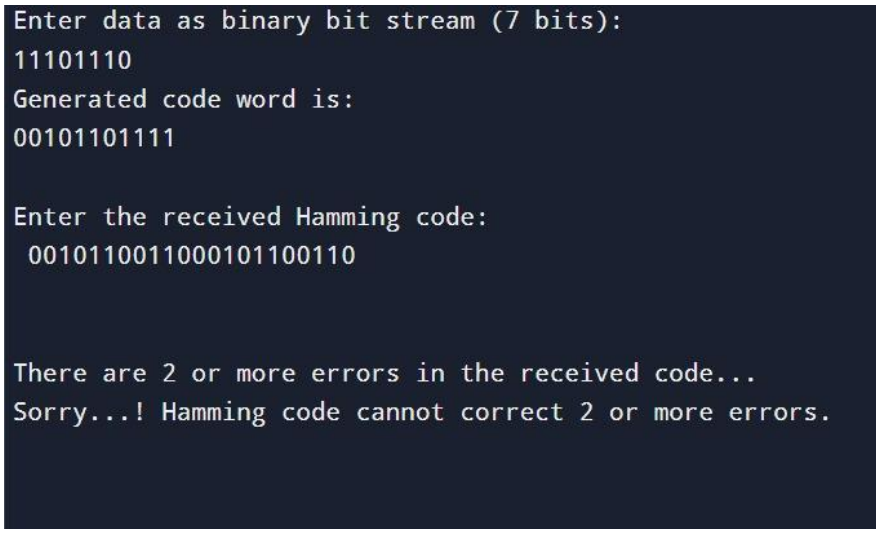
ALGORITHM:

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

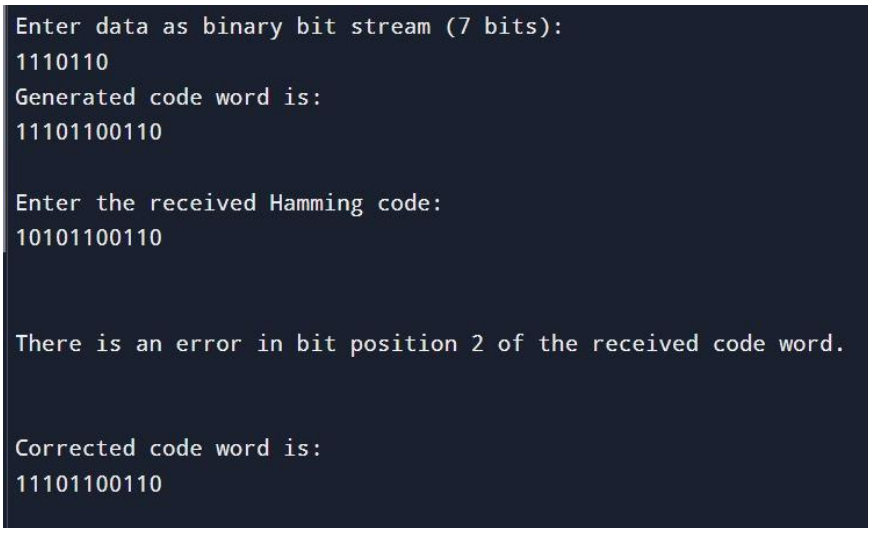
Code:

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main() {
    int i, j, k, count, err_pos = 0, flag = 0;
    char dw[20], cw[20], data[20];

    printf("Enter data as binary bit stream (7 bits):\n");
    scanf("%s", data);

    // Generating the codeword with parity bits
    for (i = 1, j = 0, k = 0; i < 12; i++) {
        if (i == (int)pow(2, j)) {
            dw[i] = '?';
            j++;
        } else {
            dw[i] = data[k];
            k++;
        }
    }

    // Calculating parity bits
    for (i = 0; i < 4; i++) {
        count = 0;
        for (j = (int)pow(2, i); j < 12; j += (int)pow(2, i) * 2) {
            for (k = 0; k < (int)pow(2, i) && j + k < 12; k++) {
                if (dw[j + k] == '1') count++;
            }
        }
        dw[(int)pow(2, i)] = (count % 2 == 0) ? '0' : '1';
    }

    // Printing the generated codeword
    printf("Generated code word is:\n");
    for (i = 1; i < 12; i++) {
        printf("%c", dw[i]);
    }
    printf("\n\nEnter the received Hamming code:\n");
    scanf("%s", cw);

    // Adjusting the index for 1-based numbering
    for (i = 12; i > 0; i--) {
        cw[i] = cw[i - 1];
    }

    // Error detection
    for (i = 0; i < 4; i++) {
        count = 0;
        for (j = (int)pow(2, i); j < 12; j += (int)pow(2, i) * 2) {
            for (k = 0; k < (int)pow(2, i) && j + k < 12; k++) {
                if (cw[j + k] == '1') count++;
            }
        }
        if (count % 2 != 0) {
            err_pos = err_pos + (int)pow(2, i);
        }
    }

    // Error correction
    if (err_pos == 0) {
        printf("\n\nThere is no error in the received code word.\n");
    } else {
        if (cw[err_pos] == dw[err_pos]) {
            printf("\n\nThere are 2 or more errors in the received code...\n");
            printf("Sorry...! Hamming code cannot correct 2 or more errors.\n");
            flag = 1;
        } else {
            printf("\n\nThere is an error in bit position %d of the received code word.\n", err_pos);
            if (flag == 0) {
                cw[err_pos] = (cw[err_pos] == '1') ? '0' : '1';
                printf("\n\nCorrected code word is:\n");
                for (i = 1; i < 12; i++) {
                    printf("%c", cw[i]);
                }
            }
        }
    }
    printf("\n\n");

    return 0;
}
```

OUTPUT:



```
Enter data as binary bit stream (7 bits):
11101110
Generated code word is:
00101101111

Enter the received Hamming code:
 00101100110001011000110



There are 2 or more errors in the received code...
Sorry...! Hamming code cannot correct 2 or more errors.
```



```
Enter data as binary bit stream (7 bits):
1110110
Generated code word is:
11101100110

Enter the received Hamming code:
10101100110


There is an error in bit position 2 of the received code word.


Corrected code word is:
11101100110
```

RESULT:

Thus the simulation of error correction code (like CRC) has been executed successfully and output got verified.