EASWARI ENGINEERING COLLEGE

(Autonomous)
Bharathi Salai, Ramapuram, Chennai 600 089.

Department : COMPUTER SCIENCE ENGINEERING

Laboratory : COMPUTER NETWORKS LABORATORY

Name	:	M. GOWTHAM . RAM
Roll No.	:	310618104032
Semester	:	V
Branch	:	CSE
Subject	:	COMPUTERNETWORKS LAB

EASWARI ENGINEERING COLLEGE

(Autonomous)
Bharathi Salai, Ramapuram, Chennai 600 089.

Department : COMPU TER SCIENCE ENGINEERING

PRACTICAL EXAMINATIONS DEC/2020 (Month / Year)

BONAFIDE CERTIFICATE

This is to certify that this practical work titled CS8581
(code)

NETWORKS LABORATORY
(Name of the Laboratory)

is the bonafide work of Mr./~~Miss~~ GOWTHAM RAM.M
(Name of the Student)

with Register Number 310618104032 in

Semester FIVE of 3rd Year in the Department of

COMPUTER SCIENCE AND ENGINEERING during the

academic year 20.20 –20 .21

Faculty Incharge **Head of the Department**

Submitted for Practical Examination held on 29/12/2020 at Easwari Engineering College, Ramapuram, Chennai – 89.

Internal Examiner **External Examiner**

INDEX :

.

EX NO: 1

Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol analyzer and examine.

Aim:

To learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and traceroute PDUs using a network protocol analyzer and examine.

Description: TCPDUMP

Tcpdump command is a famous network packet analysing tool that is used to display TCP\IP & other network packets being transmitted over the network attached to the system on which tcpdump have been installed. Tcpdump uses libpcap library to capture the network packets & is available on almost all Linux/Unix flavors.

Tcpdump command can read the contents from a network interface or from a previously created packet file or we can also write the packets to a file to be used for later. One must use the tcpdump command as root or as a user with sudo privileges

Inatalling Tcpdunp:

```
$ apt-get install tcpdum
```

1. **Get packets from all interfaces**

To get the network packets from all network interfaces, run the following command,

```
$ tcpdump -i any
```

2. **Get packets from a single interfaces**

To get the network packets from a single interface, use

```
$ tcpdump -i eth0
```

3. **Writing captured packets to file**

To write all the captured packets to a file, use the '-w' option,

```
$ tcpdump -i eth1 -w packets_file
```

4. **Reading an old tcpdump file**

To read an already created, old tcpdump file, use the following command,

```
$ tcpdump -r packets_file
```

5. **Getting more packets information with readable timestamps**

To get more information regarding the packets along with readable timestamp, use

$ tcpdump -ttttnnvvS

6. **Check packets of whole network**

To get the packets for whole network, execute the following command from terminal

$ tcpdump net 192.168.1.0/24

7. **Check packets based on IP address**

Get all the packets based on the IP address, whether source or destination or both, using the following command,

$ tcpdump host 192.168.1.100

To get packets based on source or destination of an IP address, use

$ tcpdump src 192.168.1.100

$ tcpdump dst 192.168.1.100

8. **Check packets for a protocol or port number**

To check all the packets used based on the protocol, run the following command

$ tcpdump ssh

To get packets for a single port ot for a range of ports, use

$ tcpdump port 22

$ tcpdump portrange 22-125

We can also use **'src'** & **'dst'** options to get packets for ports based on source & destination.

We can also combine two conditions with AND (and, &&), OR (or. ||) & EXCEPT (not,!). This helps when we have analyze network packets based on the some conditions.

9. **Using AND:**
   We can use 'and' or symbol '&&' to combine two conditions or mote with tcpdump. An example would be,

$ tcpdump src 192.168.1.100 && port 22 -w ssh_packets

10. **Using OR**

OR will check the command agtcpdump -i eth0 src port not 22ainst one the mentioned conditions in the command, like

$ tcpdump src 192.168.1.100 or dst 192.168.1.50 && port 22 -w ssh_packets

$ tcpdump port 443 or 80 -w http_packets

11. Using EXCEPT

EXCEPT will be used when we want not fulfill a condition, like

$ tcpdump -i eth0 src port not 22 This will monitor all the traffic on eth0 but will not capture port 22.

Description: NETSTAT

Netstat is a command line utility that tells us about all the tcp/udp/unix socket connections on our system. It provides list of all connections that are currently established or are in waiting state. This tool is extremely useful in identifying the port numbers on which an application is working and we can also make sure if an application is working or not on the port it is supposed to work.Netstat command also displays various other network related information such as routing tables, interface statistics, masquerade connections, multicast memberships etc.,

1- Checking all connections

To list out all the connections on a system, we can use 'a' option with netstat command,

$ netstat -a

This will produce all tcp, udp & unix connections from the system.

2- Checking all tcp or udp or unix socket connections

To list only the tcp connections our system, use 't' options with netstat,

$ netstat -at

Similarly to list out only the udp connections on our system, we can use 'u' option with netstat,

$ netstat -au

To only list out UNIX socket connections, we can use 'x' options,

$ netstat -ax

3- List process id/Process Name with

To get list of all connections along with PID or process name, we can use 'p' option & it can be used in combination with any other netstat option,

$ netstat -ap

4- List only port number & not the name

To speed up our output, we can use ‘n’ option as it will perform any reverse lookup & produce output with only numbers. Since no lookup is performed, our output will much faster.

```
$ netstat -an
```

5- Print only listening ports

To print only the listening ports, we will use ‘l’ option with netstat. It will not be used with ‘a’ as it prints all ports,

```
$ netstat -l
```

6- Print network stats

To print network statistics of each protocol like packet received or transmitted, we can use‘s’ options with netstat,

```
$ netstat -s
```

***7*- Print interfaces stats**

To display only the statistics on network interfaces, use ‘I’ option,

```
$ netstat -i
```

8-Display multicast group information

With option ‘g’, we can print the multicast group information for IPV4 & IPV6,

```
$ netstat -g
```

9- Display the network routing information

To print the network routing information, use ‘r’ option,

```
$ netstat -r
```

10- Continuous output

To get continuous output of netstat, use ‘c’ option

$ netstat -c

11- Filtering a single port

To filter a single port connections, we can combine ‘grep’ command with netstat,

$ netstat -anp | grep 3306

12- Count number of connections

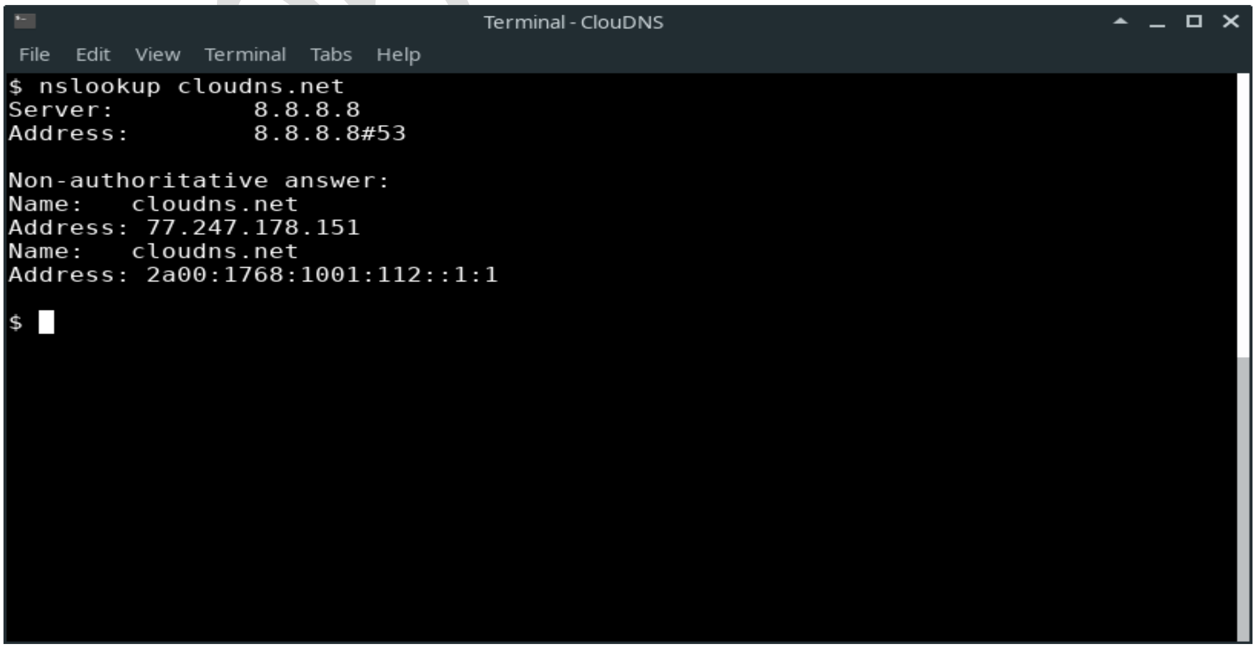
To count the number of connections from port, we can further add ‘wc’ command with netstat & grep command,

Description: NSLOOKUP commands

Nslookup – it is a powerful network administration command-line tool, available for many of the popular computer operating systems for querying Domain Name System (DNS) to obtain domain names or IP addresses, mapping or for any other specific DNS Records.

1. How to find the A record of a domain.

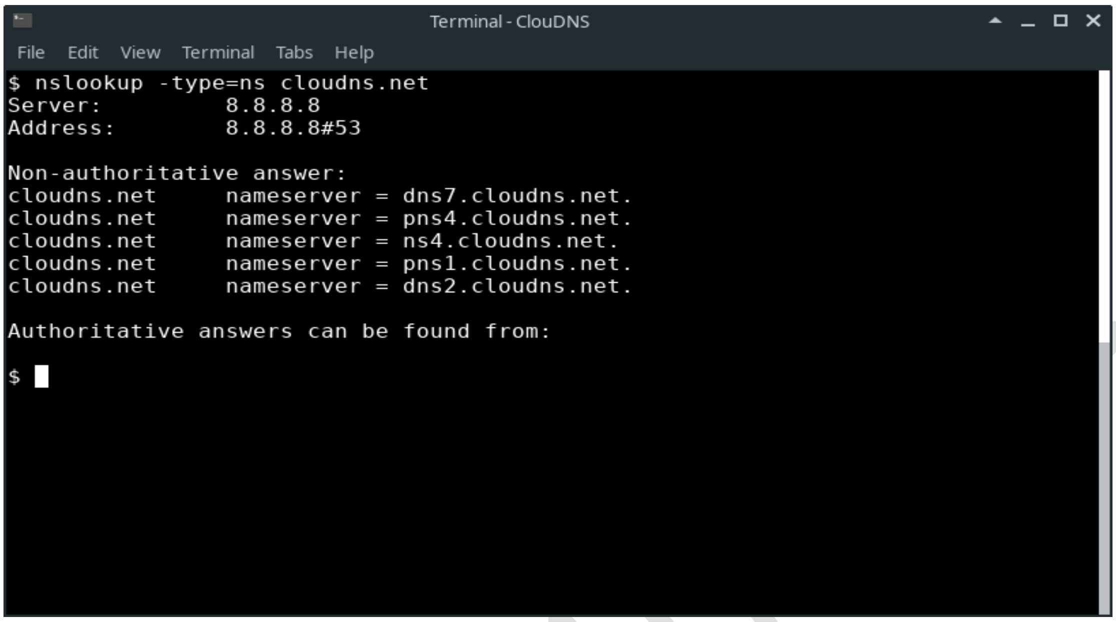
$ nslookup example.com



```
$ nslookup cloudns.net
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:    cloudns.net
Address: 77.247.178.151
Name:    cloudns.net
Address: 2a00:1768:1001:112::1:1

$ 
```

2. How to check the NS records of a domain.

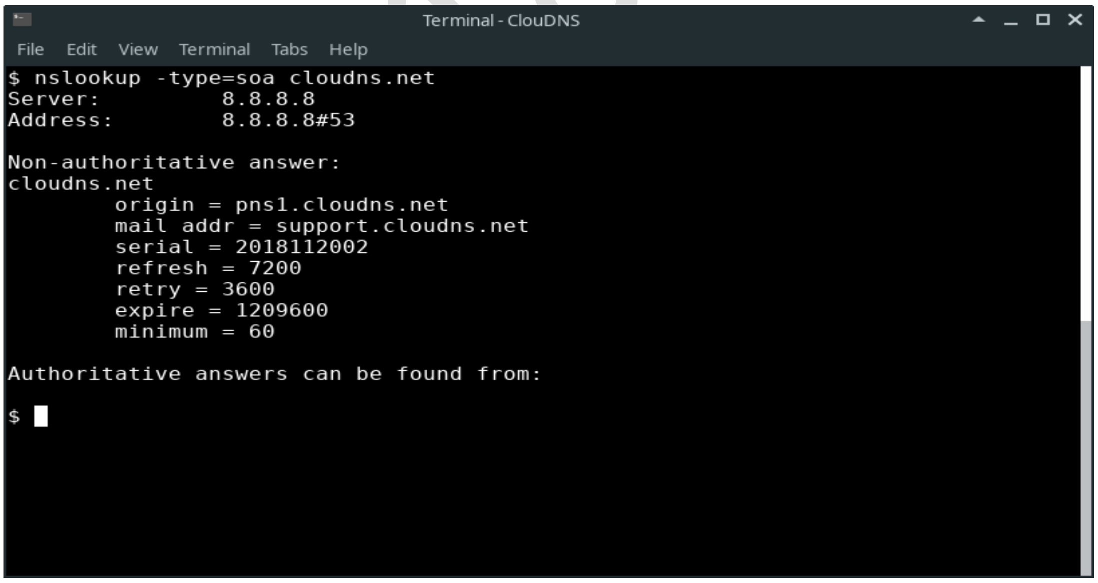
$nslookup -type=ns example.com



```
$ nslookup -type=ns cloudns.net
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
cloudns.net     nameserver = dns7.cloudns.net.
cloudns.net     nameserver = pns4.cloudns.net.
cloudns.net     nameserver = ns4.cloudns.net.
cloudns.net     nameserver = pns1.cloudns.net.
cloudns.net     nameserver = dns2.cloudns.net.

Authoritative answers can be found from:

$
```

3. How to query the SOA record of a domain.

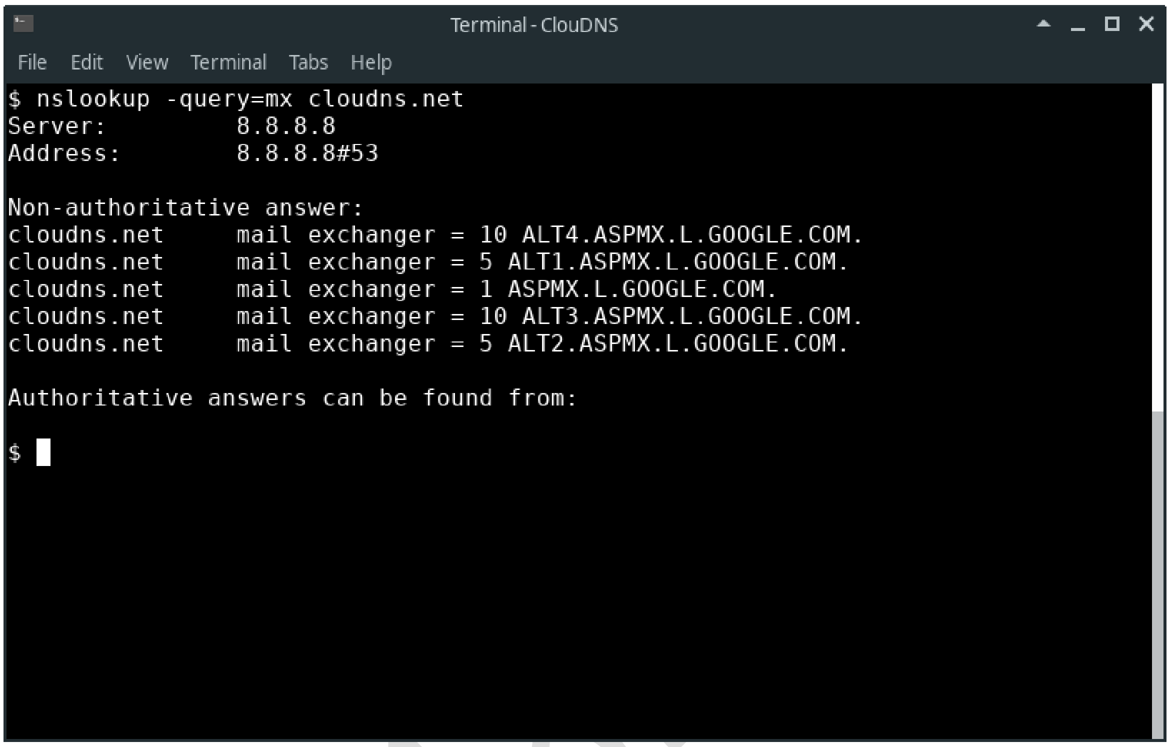
$nslookup -type=soa example.com



```
$ nslookup -type=soa cloudns.net
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
cloudns.net
        origin = pns1.cloudns.net
        mail addr = support.cloudns.net
        serial = 2018112002
        refresh = 7200
        retry = 3600
        expire = 1209600
        minimum = 60

Authoritative answers can be found from:

$
```

4. How to find the MX records responsible for the email exchange.

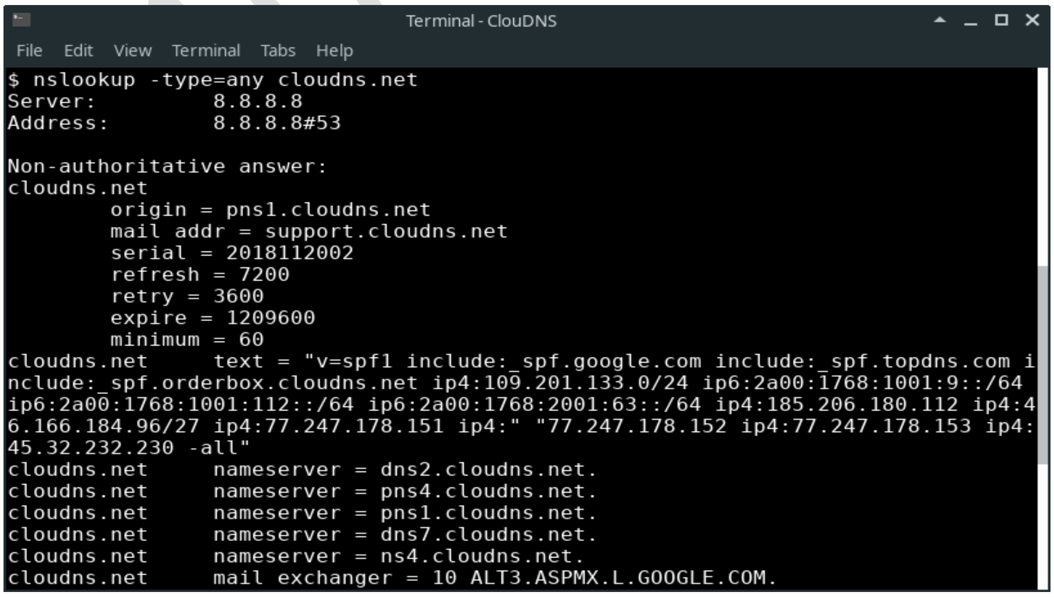
$ nslookup -query=mx example.com



```
$ nslookup -query=mx cloudns.net
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
cloudns.net     mail exchanger = 10 ALT4.ASPMX.L.GOOGLE.COM.
cloudns.net     mail exchanger = 5 ALT1.ASPMX.L.GOOGLE.COM.
cloudns.net     mail exchanger = 1 ASPMX.L.GOOGLE.COM.
cloudns.net     mail exchanger = 10 ALT3.ASPMX.L.GOOGLE.COM.
cloudns.net     mail exchanger = 5 ALT2.ASPMX.L.GOOGLE.COM.

Authoritative answers can be found from:

$
```

5. How to find all of the available DNS records of a domain.

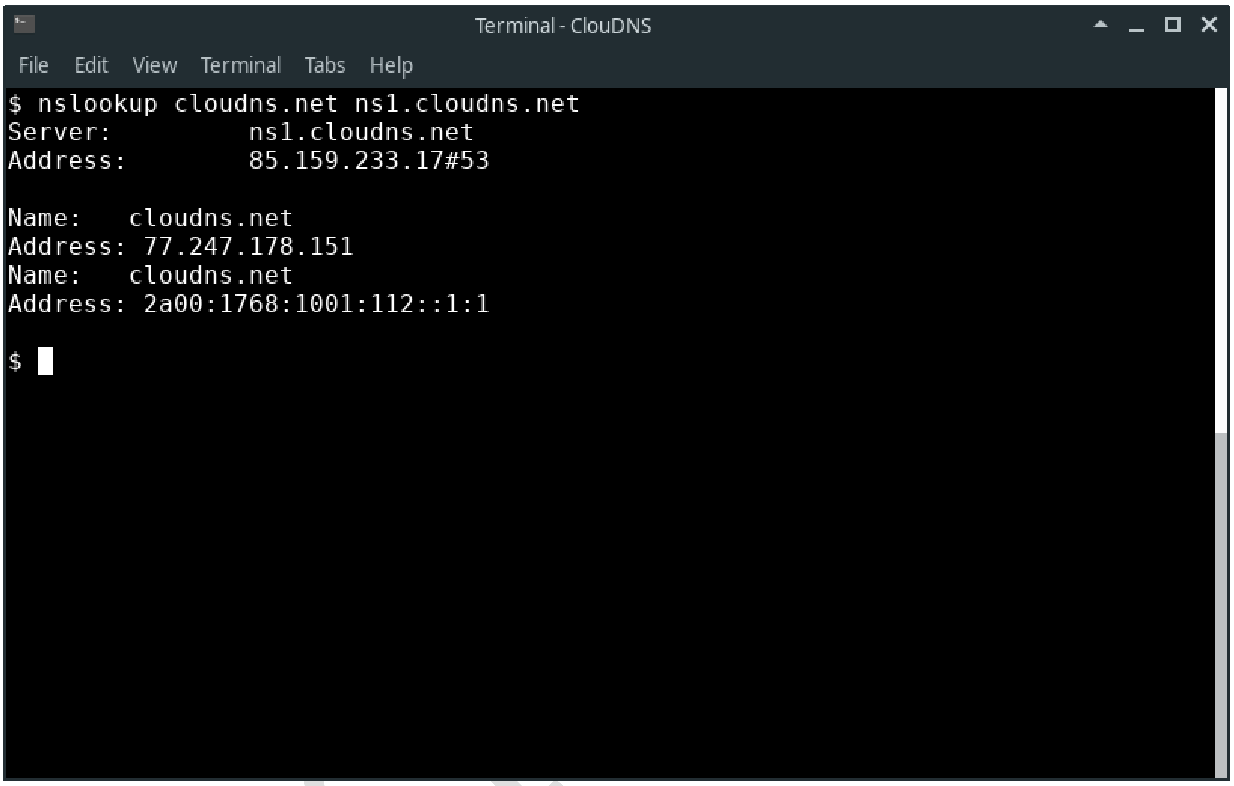
$ nslookup -type=any example.com



```
$ nslookup -type=any cloudns.net
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
cloudns.net
        origin = pns1.cloudns.net
        mail addr = support.cloudns.net
        serial = 2018112002
        refresh = 7200
        retry = 3600
        expire = 1209600
        minimum = 60
cloudns.net     text = "v=spf1 include:_spf.google.com include:_spf.topdns.com i
nclude:_spf.orderbox.cloudns.net ip4:109.201.133.0/24 ip6:2a00:1768:1001:9::/64
ip6:2a00:1768:1001:112::/64 ip6:2a00:1768:2001:63::/64 ip4:185.206.180.112 ip4:4
6.166.184.96/27 ip4:77.247.178.151 ip4:" "77.247.178.152 ip4:77.247.178.153 ip4:
45.32.232.230 -all"
cloudns.net     nameserver = dns2.cloudns.net.
cloudns.net     nameserver = pns4.cloudns.net.
cloudns.net     nameserver = pns1.cloudns.net.
cloudns.net     nameserver = dns7.cloudns.net.
cloudns.net     nameserver = ns4.cloudns.net.
cloudns.net     mail exchanger = 10 ALT3.ASPMX.L.GOOGLE.COM.
```

6. How to check the using of a specific DNS Server.

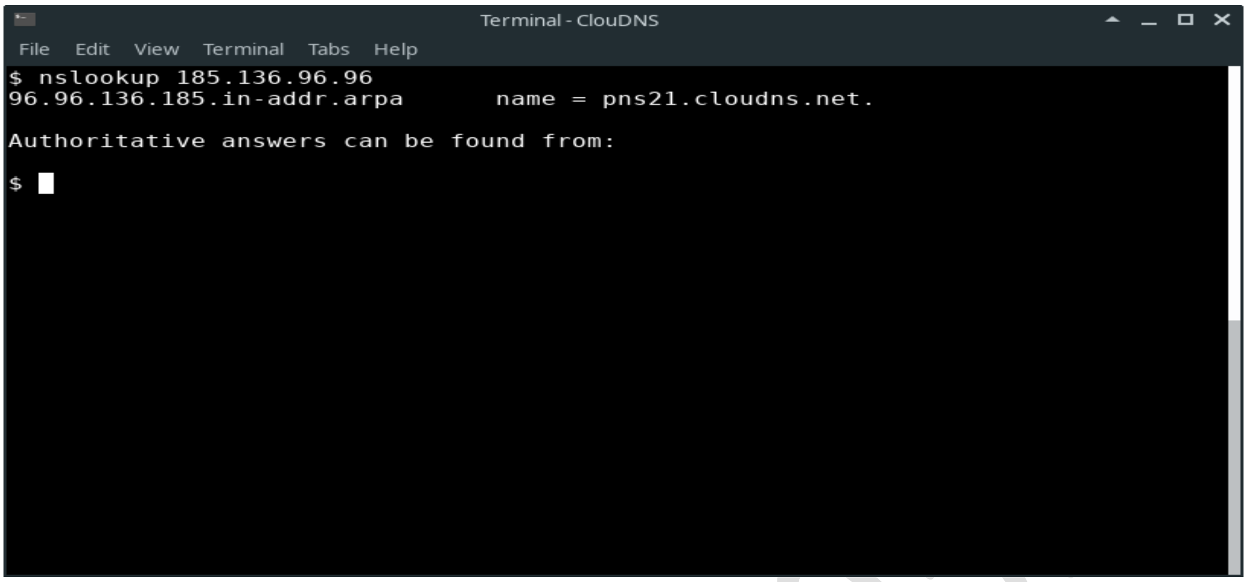
$ nslookup example.com ns1.nsexample.com



```
$ nslookup cloudns.net ns1.cloudns.net
Server:         ns1.cloudns.net
Address:        85.159.233.17#53

Name:   cloudns.net
Address: 77.247.178.151
Name:   cloudns.net
Address: 2a00:1768:1001:112::1:1

$ 
```

7. How to check the Reverse DNS Lookup.

$ nslookup 10.20.30.40

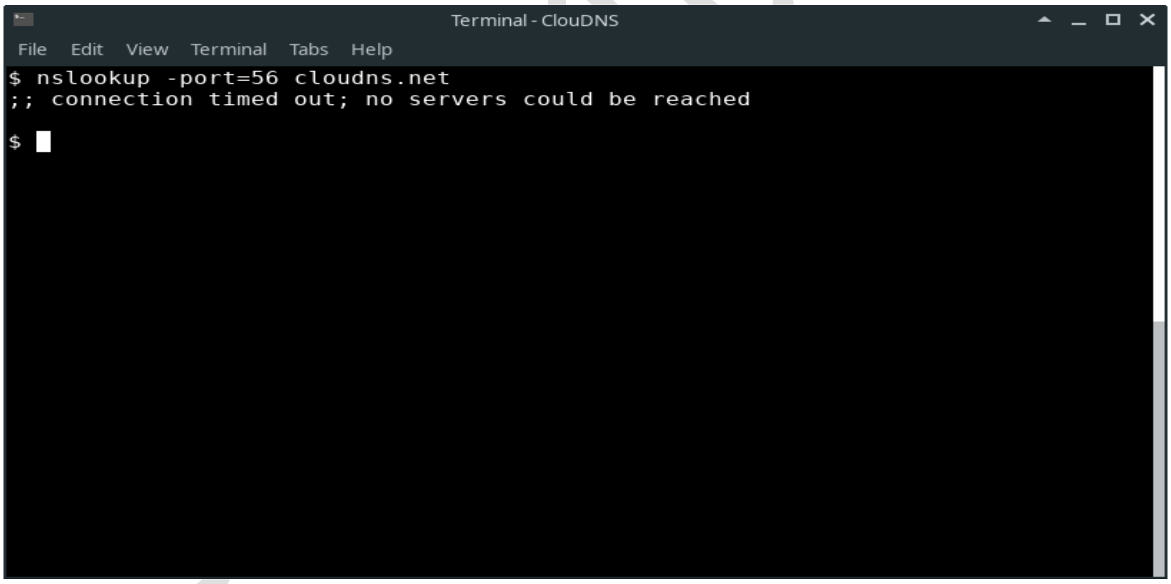


```
$ nslookup 185.136.96.96
96.96.136.185.in-addr.arpa      name = pns21.cloudns.net.

Authoritative answers can be found from:

$ 
```

---

8. How to change the port number for the connection.

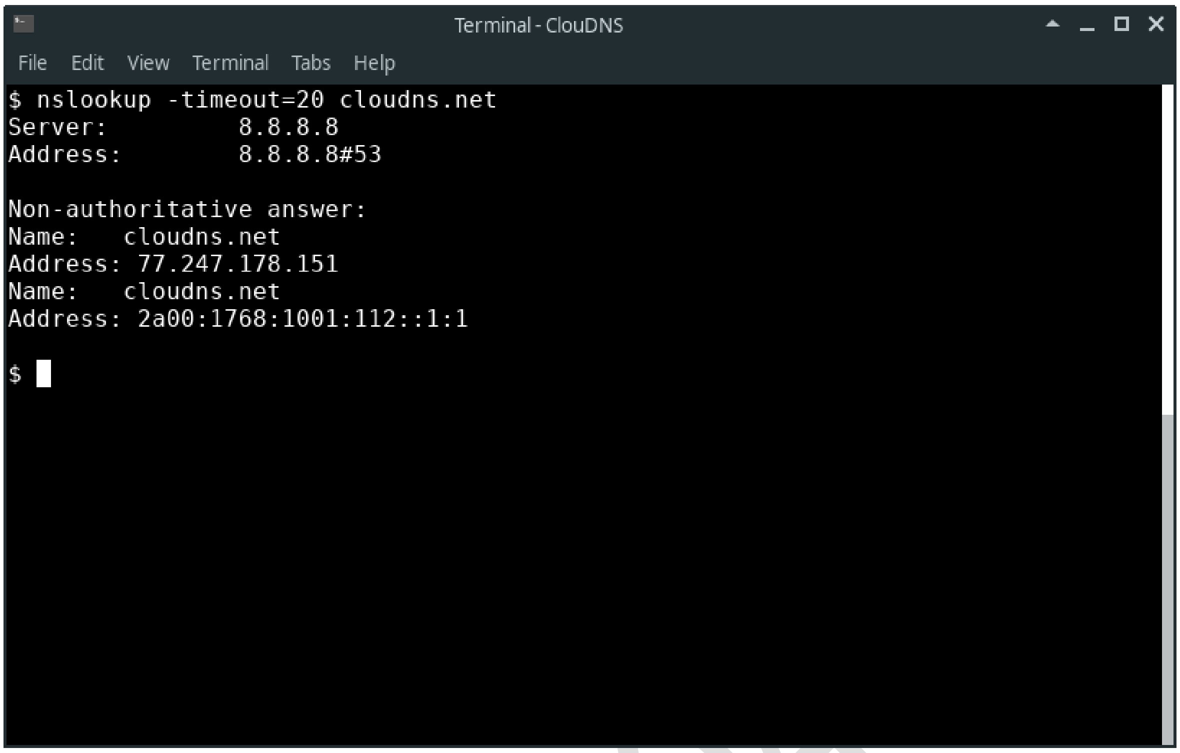
$ nslookup -port=56 example.com



```
$ nslookup -port=56 cloudns.net
;; connection timed out; no servers could be reached

$ 
```

9. How to change the timeout interval for a reply.

$ nslookup -timeout=20 example.com

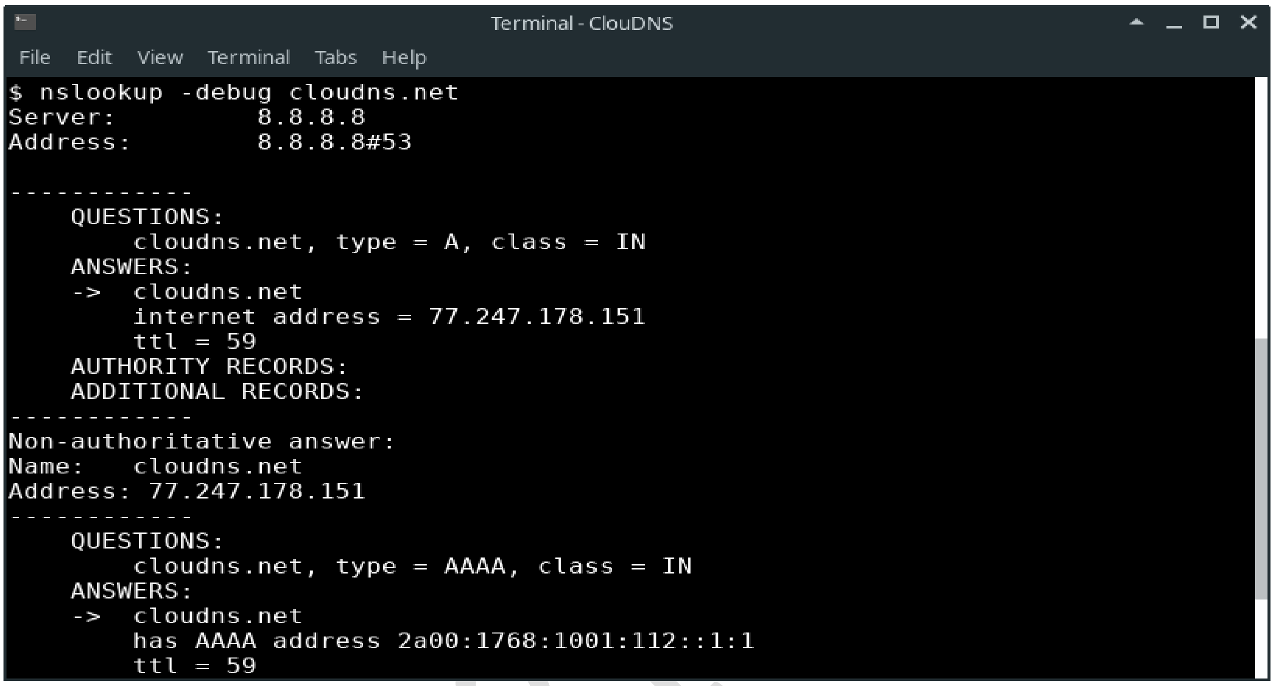


```
$ nslookup -timeout=20 cloudns.net
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
Name:   cloudns.net
Address: 77.247.178.151
Name:   cloudns.net
Address: 2a00:1768:1001:112::1:1

$ 
```

10. How to enable debug mode.

Debug mode provides important and detailed information both for the question and for the received answer.

$ nslookup -debug example.com



```
$ nslookup -debug cloudns.net
Server:         8.8.8.8
Address:        8.8.8.8#53

------------
    QUESTIONS:
        cloudns.net, type = A, class = IN
    ANSWERS:
    ->  cloudns.net
        internet address = 77.247.178.151
        ttl = 59
    AUTHORITY RECORDS:
    ADDITIONAL RECORDS:
------------
Non-authoritative answer:
Name:   cloudns.net
Address: 77.247.178.151
------------
    QUESTIONS:
        cloudns.net, type = AAAA, class = IN
    ANSWERS:
    ->  cloudns.net
        has AAAA address 2a00:1768:1001:112::1:1
        ttl = 59
```

Result:

Thus the learning of commands like tcpdump, netstat, ifconfig, nslookup and traceroute has done. Capture ping and traceroute PDUs using a network protocol analyzer and examined.

EX NO: 2 HTTP web client program to download a web page using TCP sockets

Aim:

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

ALGORITHM

Server

1. Include necessary header files to support functions for Socket definitions, Socket Types, Internet
   addresses, I/O functions, Unix system calls.
2. Declare variables for Socket ID, Port number, Socket addresses, buffer, etc.
3. Create Socket for server.
4. Bind socket with addresses.
5. Specify number of allowed connections.
6. Wait for connection.
7. Accept connection (If any).
8. Repeat the steps 8and 9 until the socket is closed.
9. Read the requested file to be transmitted from the client
10. The requested file is transferred by write to the new location.

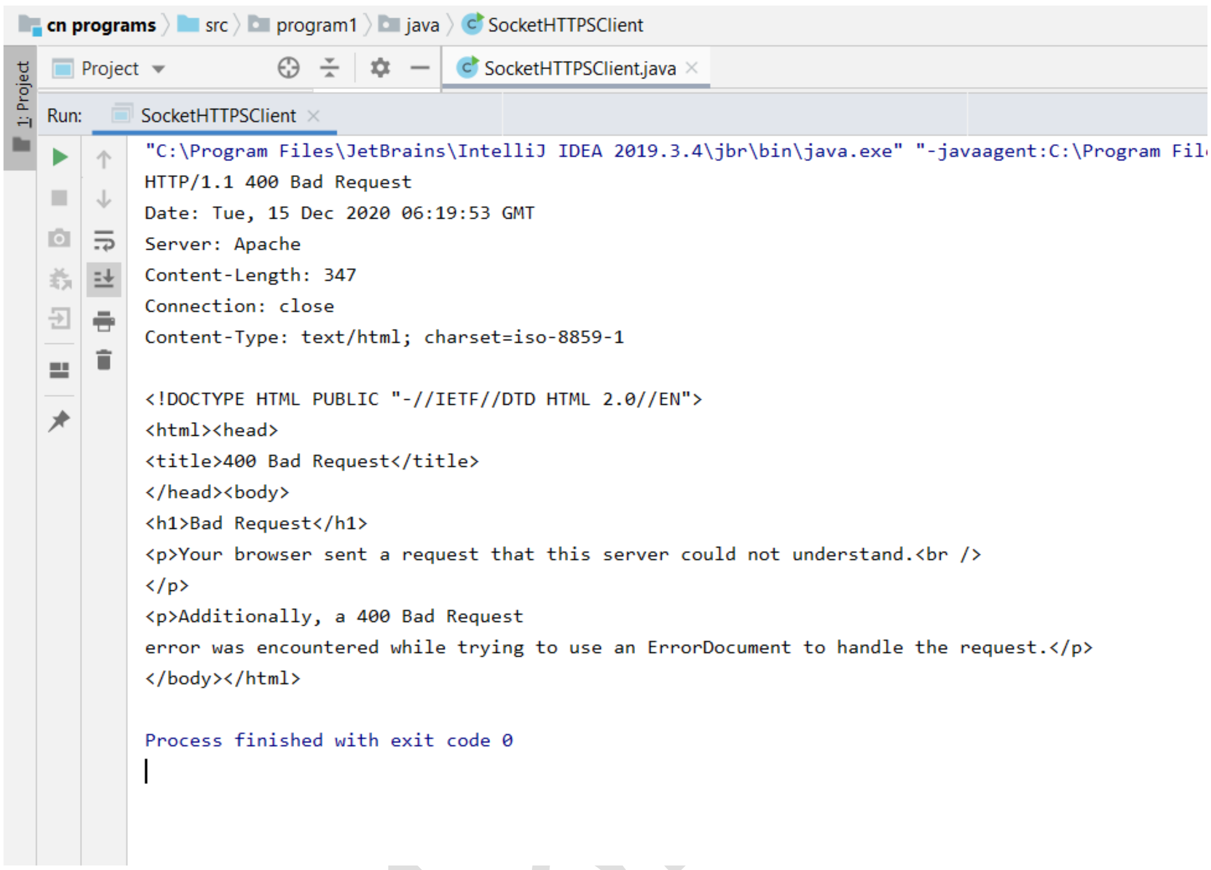
Client

1. Include necessary header files to support functions for Socket definitions, Socket types, Internet
   addresses, I/O functions, Unix system calls.
2. Declare variables for Socket ID, Port number, Socket addresses, Character buffer, etc.
3. Create Socket for Client.
4. Connect client socket to the server socket addresses.
5. Repeat the steps 8and 9 until the socket is closed.
6. Send file name to the Connected Socket
7. Retrieve information from Connected Socket and display it.

Program:

```
package program1.java;
import java.io.*;
import java.net.*;

public class SocketHTTPSClient {
   public static void main(String[] args) {

      String hostName = "www.gowtham.com";
      int portNumber = 80;

      try {
         Socket socket = new Socket(hostName, portNumber);
         PrintWriter out =
               new PrintWriter(socket.getOutputStream(), true);
         BufferedReader in =
               new BufferedReader(
                     new InputStreamReader(socket.getInputStream()));
         out.println("GET / HTTP/1.1\nHost: www.gowtham.com\n\n");
         String inputLine;
         while ((inputLine = in.readLine()) != null) {
            System.out.println(inputLine);
         }
      } catch (UnknownHostException e) {
         System.err.println("Don't know about host " + hostName);
         System.exit(1);
      } catch (IOException e) {
         System.err.println("Couldn't get I/O for the connection to " +
               hostName);
         System.exit(1);
      }
   }
}
```

output:



```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.4\jbr\bin\java.exe" "-javaagent:C:\Program Fil
HTTP/1.1 400 Bad Request
Date: Tue, 15 Dec 2020 06:19:53 GMT
Server: Apache
Content-Length: 347
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<p>Additionally, a 400 Bad Request
error was encountered while trying to use an ErrorDocument to handle the request.</p>
</body></html>

Process finished with exit code 0
```

Result:

Thus the program for HTTP web client to download web page using TCP sockets was implemented and output was verified.

EX No:3 Perform Applications using TCP sockets like:
a) Echo client and echo server
b) Chat
c) File Transfer

a) **Echo client and echo server**:

Aim:

To write Applications using TCP sockets like Echo client and echo server. This is an echo server: the server that echoes back all data it receives to a client that sent it.

Algorithm

Server

1. A TCP socket is created.
2. An Internet socket address structure is filled in with the wildcard address (INADDR_ANY) and the server's well-known port (PORT).
3. The socket is converted into a listening socket by calling the listen function.
4. The server blocks in the call to accept, waiting for the client connection to complete.
5. When the connection is established, the server reads the line from the client using readn and echoes it back to the client using writen.
6. Finally, the server closes the connected socket.

Client:

1. A TCP socket is created.
2. An Internet socket address structure is filled in with the server's IP address and the same port number.
3. The connect function establishes the connection with the server.
4. The client reads a line of text from the standard input using fgets, writes it to the server using writen, reads back the server's echo of the line using readline and outputs the echoed line to the standard output using fputs.

Program:

SERVER.PY:

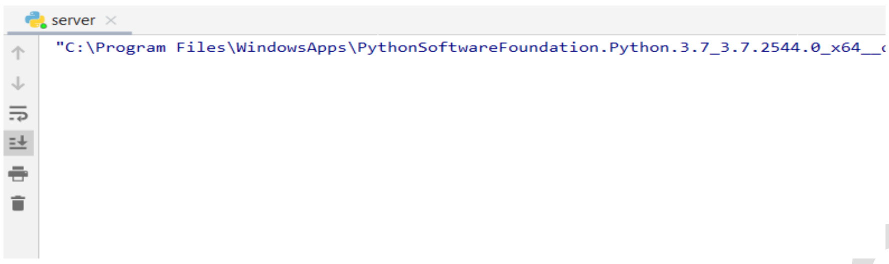
```
import socket

host = ''
port = 12345
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((host, port))
s.listen(1)
conn, addr = s.accept()
print('Connected by', addr)
while True:
    data = conn.recv(1024)
    if not data: break
    conn.sendall(data)
conn.close()
```

CLIENT.PY:

```
import socket

host = socket.gethostname()
port = 12345

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
s.sendall(b'Hi, this is Gowtham')
data = s.recv(1024)
s.close()
print('Received', data)
```

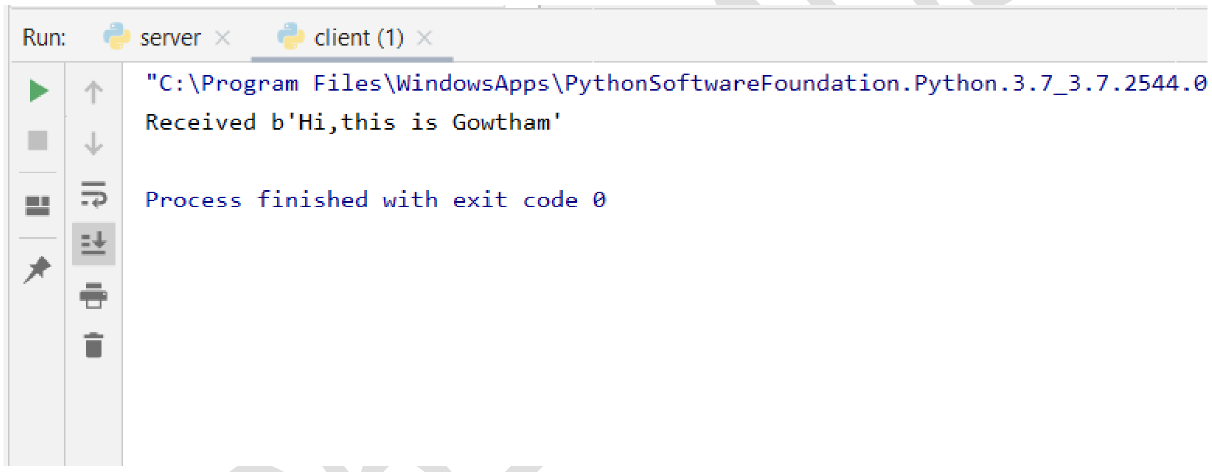
Output:

Server.py wont have any putput:



```
"C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.7_3.7.2544.0_x64__
```

Client.py:



```
"C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.7_3.7.2544.0
Received b'Hi,this is Gowtham'

Process finished with exit code 0
```

Result:

Thus the Applications using TCP sockets like Echo client and echo server has been done.

b) Chat

Aim

To perform the full duplex chat by sending and receiving the message from the client to server and vice versa using TCP sockets.

Algorithm

Server

1. A TCP socket is created.
2. An Internet socket address structure is filled in with the wildcard address (INADDR_ANY) and the server’s well-known port (PORT).
3. The socket is converted into a listening socket by calling the listen function.
4. The server blocks in the call to accept, waiting for the client connection to complete.
5. When the connection is established, the server reads the line from the client using connected socket and display the message in the standard output using fputs.
6. Then again the server reads a line of text from the standard input and writes it back to the client through the connected socket.
7. The server went through the steps (5) and (6) until it receives 'bye' either from the standard input or client.
8. Finally, the server closes the connected socket.

Client:

1. A TCP socket is created.
2. An Internet socket address structure is filled in with the server’s IP address and the same port number.
3. The connect function establishes the connection with the server.
4. When the connection is established, the client reads the line from the standard input using fgets and send the message to the server through the socket.
5. Then again the client reads a line of text from the server through the connected socket and writes it back to the standard output using fputs.
6. The client went through the steps (4) and (5) until it receives 'bye' either from the standard input or server.
7. Finally, the client closes the connected socket.

program:

Server.py:

```
import socket
import select

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 1234


server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)


server_socket.bind((IP, PORT))

server_socket.listen()


sockets_list = [server_socket]

clients = {}

print(f'Listening for connections on {IP}:{PORT}...')


def receive_message(client_socket):

    try:

        message_header = client_socket.recv(HEADER_LENGTH)

        if not len(message_header):
            return False


        message_length = int(message_header.decode('utf-8').strip())
        return {'header': message_header, 'data':
client_socket.recv(message_length)}

    except:

        return False

while True:


    read_sockets, _, exception_sockets = select.select(sockets_list, [], sockets_list)



    for notified_socket in read_sockets:

        if notified_socket == server_socket:


            client_socket, client_address = server_socket.accept()
            user = receive_message(client_socket)
            if user is False:
                continue


            sockets_list.append(client_socket)


            clients[client_socket] = user

            print('Accepted new connection from {}:{}, username:
{}'.format(*client_address, user['data'].decode('utf-8')))

        else:


            message = receive_message(notified_socket)

            if message is False:
                print('Closed connection from:
{}'.format(clients[notified_socket]['data'].decode('utf-8')))

                sockets_list.remove(notified_socket)
                del clients[notified_socket]

                continue


            user = clients[notified_socket]

            print(f'Received message from {user["data"].decode("utf-8")}:
{message["data"].decode("utf-8")}')


            for client_socket in clients:


                client_socket.send(user['header'] + user['data'] + message['header'] +
message['data'])


    for notified_socket in exception_sockets:


        sockets_list.remove(notified_socket)
        del clients[notified_socket]
```

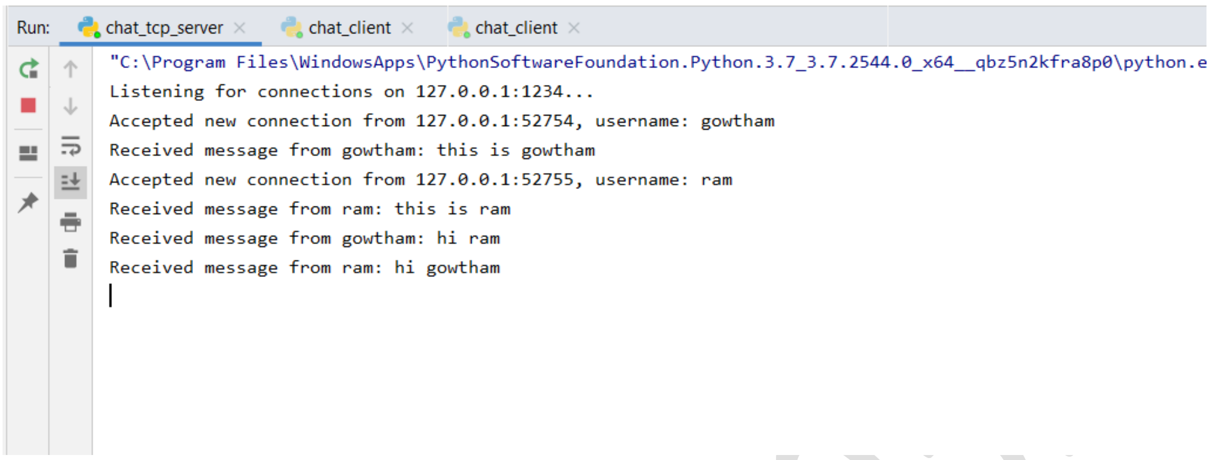
Client.py:

```
import socket
import select
import errno
import sys

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 1234
my_username = input("Username: ")

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


client_socket.connect((IP, PORT))


client_socket.setblocking(False)


username = my_username.encode('utf-8')
username_header = f"{len(username):<{HEADER_LENGTH}}".encode('utf-8')
client_socket.send(username_header + username)

while True:


    message = input(f'{my_username} > ')


    if message:


        message = message.encode('utf-8')
        message_header = f"{len(message):<{HEADER_LENGTH}}".encode('utf-8')
        client_socket.send(message_header + message)

    try:

        while True:
```

```
        username_header = client_socket.recv(HEADER_LENGTH)


        if not len(username_header):
            print('Connection closed by the server')
            sys.exit()


        username_length = int(username_header.decode('utf-8').strip())


        username = client_socket.recv(username_length).decode('utf-8')


        message_header = client_socket.recv(HEADER_LENGTH)
        message_length = int(message_header.decode('utf-8').strip())
        message = client_socket.recv(message_length).decode('utf-8')


        print(f'{username} > {message}')

    except IOError as e:

        if e.errno != errno.EAGAIN and e.errno != errno.EWOULDBLOCK:
            print('Reading error: {}'.format(str(e)))
            sys.exit()


        continue

    except Exception as e:
        print('Reading error: '.format(str(e)))
        sys.exit()
```

Output:

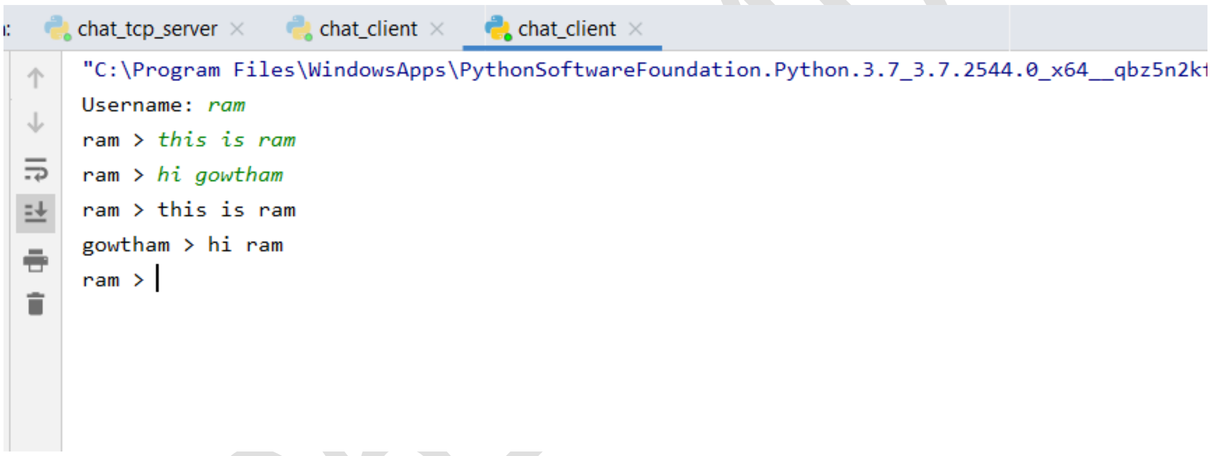
Server.py:



Run: chat_tcp_server × chat_client × chat_client ×

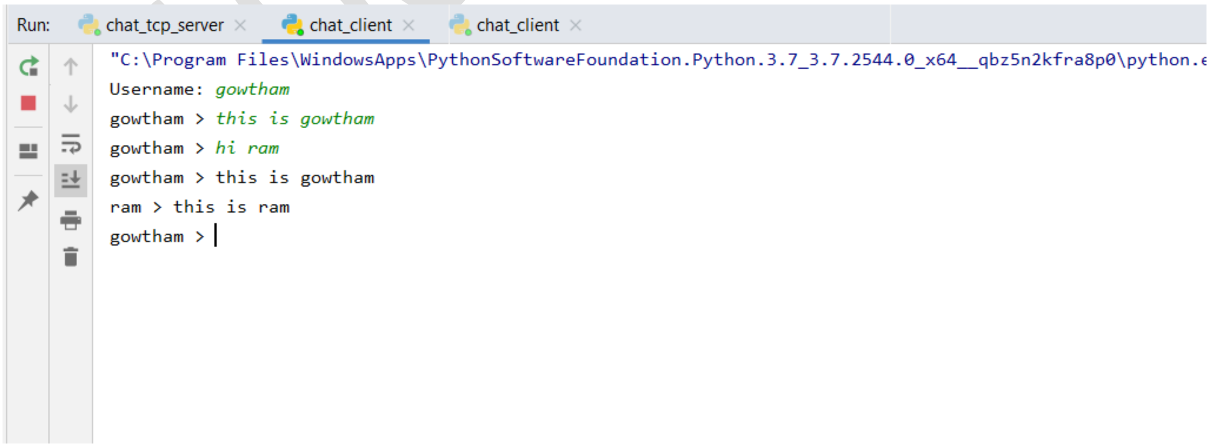
```
"C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.7_3.7.2544.0_x64__qbz5n2kfra8p0\python.e
Listening for connections on 127.0.0.1:1234...
Accepted new connection from 127.0.0.1:52754, username: gowtham
Received message from gowtham: this is gowtham
Accepted new connection from 127.0.0.1:52755, username: ram
Received message from ram: this is ram
Received message from gowtham: hi ram
Received message from ram: hi gowtham
```

Client.py:



chat_tcp_server × chat_client × chat_client ×

```
"C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.7_3.7.2544.0_x64__qbz5n2kf
Username: ram
ram > this is ram
ram > hi gowtham
ram > this is ram
gowtham > hi ram
ram >
```



Run: chat_tcp_server × chat_client × chat_client ×

```
"C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.7_3.7.2544.0_x64__qbz5n2kfra8p0\python.e
Username: gowtham
gowtham > this is gowtham
gowtham > hi ram
gowtham > this is gowtham
ram > this is ram
gowtham >
```

Result: Thus the Applications using TCP sockets like chat server has been executed and output was verified

c) File Transfer:

Aim:

To write a program for File Transfer Application using TCP socket.

Server:

Step 1: Start the program execution
Step 2: Create a socket with address family AF_INET, type SOCK_STERAM and default
protocol.
Step 3: Initialize the socket and set its attributes.
Step 4: Bind the server to socket using bind function.
Step 5: wait for the client request on request establish a connection using accept() function.
Step 6: Read the source and destination files names from the client.
Step 7: Open the source and destination files.
Step 8: Read one line of source file and send it to client.
Step 9: Receive the line back from the client.
Step 10: Repeat steps 8&9 until the end of the source file.
Step 11: close the source and destination files.
Step 12: close the connection and goto step5.

Client:
Step 1: start the program execution
Step 2: Create a socket with address family AEINET type SOCK_STREAM and default protocol.
Step 3: Initialize the socket and set its attribute set required port no.
Step 4: Connect to server using connect () function to initiate the request.
Step 5: send the source and destination file names to server.
Step 6: Receive the string from the server and print it at the console.
Step 7: send the string to the server.
Step 8: Repeat step 6&7 until the server terminates and connection.
Step 9: stop.

Program:

Server.py:

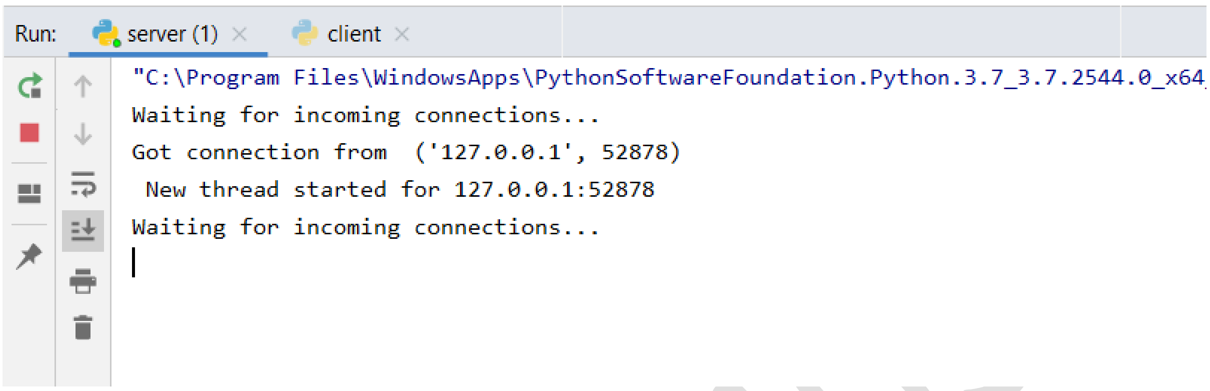
```
import socket
from threading import Thread
from Assignment3 import*

TCP_IP = 'localhost'
TCP_PORT = 9001
BUFFER_SIZE = 1024

class ClientThread(Thread):

    def __init__(self,ip,port,sock):
        Thread.__init__(self)
        self.ip = ip
        self.port = port
        self.sock = sock
        print (" New thread started for "+ip+":"+str(port))

    def run(self):
        filename='mytext.txt'
        f = open(filename,'rb')
        while True:
            l = f.read(BUFFER_SIZE)
            while (l):
                self.sock.send(l)
                #print('Sent ',repr(l))
                l = f.read(BUFFER_SIZE)
            if not l:
                f.close()
                self.sock.close()
                break

tcpsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
tcpsock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
tcpsock.bind((TCP_IP, TCP_PORT))
threads = []

while True:
    tcpsock.listen(5)
    print ("Waiting for incoming connections...")
    (conn, (ip,port)) = tcpsock.accept()
    print ('Got connection from ', (ip,port))
    newthread = ClientThread(ip,port,conn)
    newthread.start()
    threads.append(newthread)

for t in threads:
  t.join()
```

client.py:

```
import socket

TCP_IP = 'localhost'
TCP_PORT = 9001
BUFFER_SIZE = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
with open('received_file', 'wb') as f:
    print ('file opened')
    while True:
        #print('receiving data...')
        data = s.recv(BUFFER_SIZE)
        print('data=%s', (data))
        if not data:
            f.close()
            print ('file close()')
            break
        # write data to a file
        f.write(data)

print('Successfully get the file')
s.close()
print('connection closed')
```

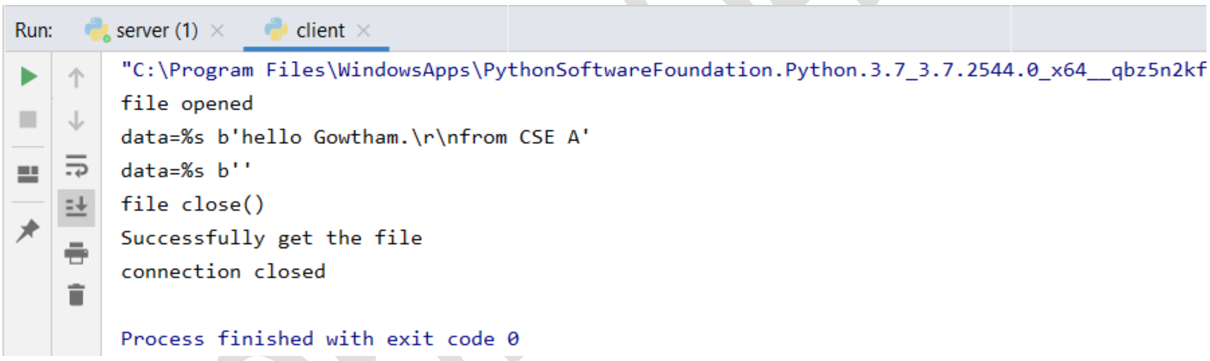
Output:

Server.py:



```
"C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.7_3.7.2544.0_x64
Waiting for incoming connections...
Got connection from  ('127.0.0.1', 52878)
 New thread started for 127.0.0.1:52878
Waiting for incoming connections...
```

Client.py:



```
"C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.7_3.7.2544.0_x64__qbz5n2kf
file opened
data=%s b'hello Gowtham.\r\nfrom CSE A'
data=%s b''
file close()
Successfully get the file
connection closed

Process finished with exit code 0
```

Result:

Thus the C program for File Transfer Application using TCP socket is executed successfully and the output was verified.

EX.NO: 4 **Simulation of DNS using UDP sockets.**

Aim

To write a program in C to simulate the Domain Name System.

Algorithm:

Server

1. A UDP socket is created.
2. An Internet socket address structure is filled in with the wildcard address (INADDR_ANY) and the server's well-known port (PORT).
3. The socket is converted into a listening socket by calling the listen function.
4. The server blocks in the call to accept, waiting for the client connection to complete.
5. When the connection is established, the server reads the domain name from the client using readn.
6. It then finds out the corresponding address using sendto() and sends it back to the client using writen.
7. Finally, the server closes the connected socket.

Client:

1. A UDP socket is created.
2. An Internet socket address structure is filled in with the server's IP address and the same port number.
3. The connect function establishes the connection with the server.
4. The client reads the domain name from the standard input using fgets, writes it to the server using writen.
5. It then reads back the server reply using recvfrom() and outputs the same to the standard output using fputs.

Program:

Server:

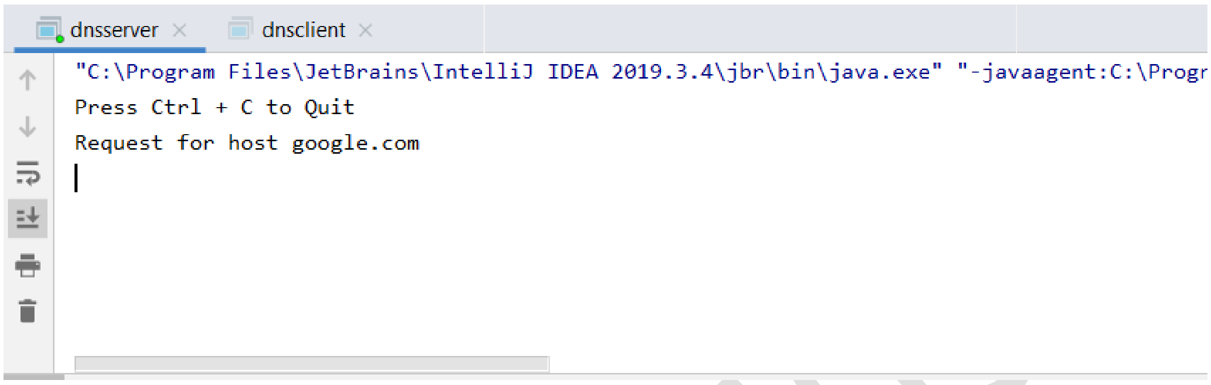
```
package program2.java;
import java.io.*;
import java.net.*;

public class dnsserver {
   private static int indexOf(String[] array, String str) {
      str = str.trim();
      for (int i=0; i < array.length; i++) {
         if (array[i].equals(str)) return i; }
      return -1;
   }
   public static void main(String arg[])throws IOException {
      String[] hosts = {"gmail.com","google.com", "facebook.com"};
      String[] ip = { "172.16.254.1", "172.217.11.5","172.217.11.14", "31.13.71.36"};
      System.out.println("Press Ctrl + C to Quit");
      while (true) {
         DatagramSocket serversocket = new DatagramSocket(1362);
         byte[] senddata = new byte[1021];
         byte[] receivedata = new byte[1021];
         DatagramPacket recvpack = new DatagramPacket(receivedata,
receivedata.length);
         serversocket.receive(recvpack);
         String sen = new String(recvpack.getData());
         InetAddress ipaddress = recvpack.getAddress();
         int port = recvpack.getPort();
         String capsent;
         System.out.println("Request for host " + sen);
         if (indexOf(hosts, sen) != -1)
            capsent = ip[indexOf(hosts, sen)];
         else
            capsent = "Host Not Found";
         senddata = capsent.getBytes();
         DatagramPacket pack = new DatagramPacket(senddata, senddata.length,
ipaddress, port);
         serversocket.send(pack);
         serversocket.close();
      }   }

}
```

client:

```
package program2.java;
import java.io.*;
import java.net.*;
public class dnsclient {
   public static void main(String args[]) throws IOException {
      BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
      DatagramSocket clientsocket = new DatagramSocket();
      InetAddress ipaddress;
      if (args.length == 0)
         ipaddress = InetAddress.getLocalHost();
      else
         ipaddress = InetAddress.getByName(args[0]);
      byte[] senddata = new byte[1024];
      byte[] receivedata = new byte[1024];
      int portaddr = 1362;
      System.out.print("Enter the hostname : ");
      String sentence = br.readLine();
      senddata = sentence.getBytes();
      DatagramPacket pack = new DatagramPacket(senddata, senddata.length,
ipaddress, portaddr);
      clientsocket.send(pack);
      DatagramPacket recvpack = new DatagramPacket(receivedata,
receivedata.length);
      clientsocket.receive(recvpack);
      String modified = new String(recvpack.getData());
      System.out.println("IP Address: " + modified);
      clientsocket.close();
   }

}
```

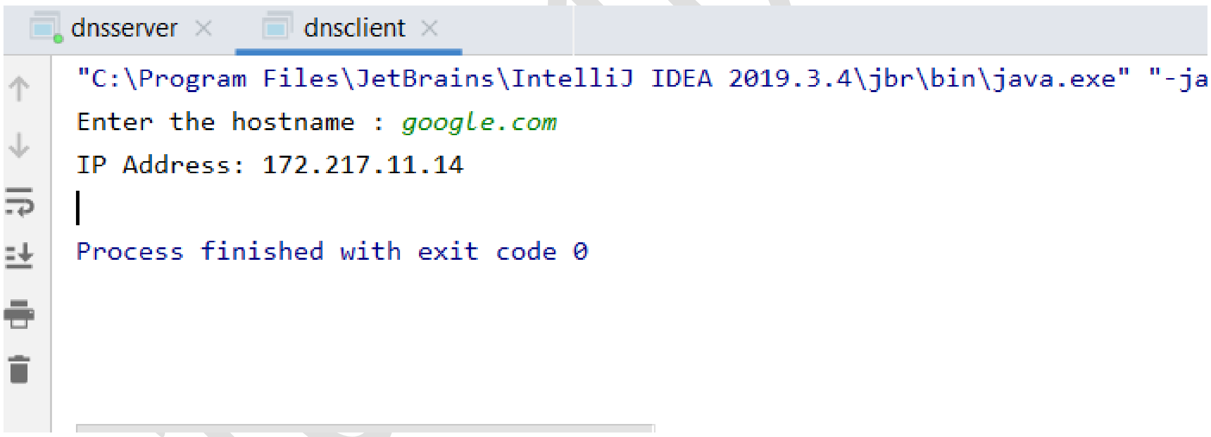
output:

server:



```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.4\jbr\bin\java.exe" "-javaagent:C:\Progr
Press Ctrl + C to Quit
Request for host google.com
```

client:



```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.4\jbr\bin\java.exe" "-ja
Enter the hostname : google.com
IP Address: 172.217.11.14

Process finished with exit code 0
```

Result:

To write a program in C to simulate the Domain Name System using UDP is executed and the output is verified successfully.

EX.NO:5 **Code simulating ARP /RARP protocols**

Aim:

To write a C program to implement Address Resolution Protocol/Reverse Address Resolution

Protocol.

ALGORITHM: ARP:

SERVER:

STEP 1: Include header files.

STEP 2: Create the socket address structure.

STEP 3: IP address, port number and family is initialized with server's socket address structure

STEP 4: Socket address is manipulated using the byte manipulation function bzero().

STEP 5: Socket created using socket() function.

STEP 6: Using bind () function, socket is bound with the server's well known port.

STEP 7: Enter number of client, accept those connection with those client with specified port.

STEP 8: The IP address is received from the client and the corresponding MAC address is sent to the client.

STEP 9: Socket is closed.

CLIENT:

STEP 1: Include header files.

STEP 2: Socket address structure is created.

STEP 3: Server's socket address structure is initialized with IP address, port number and family.

STEP 4: Server and the client's address are manipulated using the byte manipulation functionbzero().

STEP 5: Socket is created using socket () function.

STEP 6: Socket is bound with the server's well known port.

STEP 7: The IP address is sent to the server.

STEP 9: Corresponding MAC address is received from the server and it is printed.STEP 10: Socket is closed.

Program:

server arp:

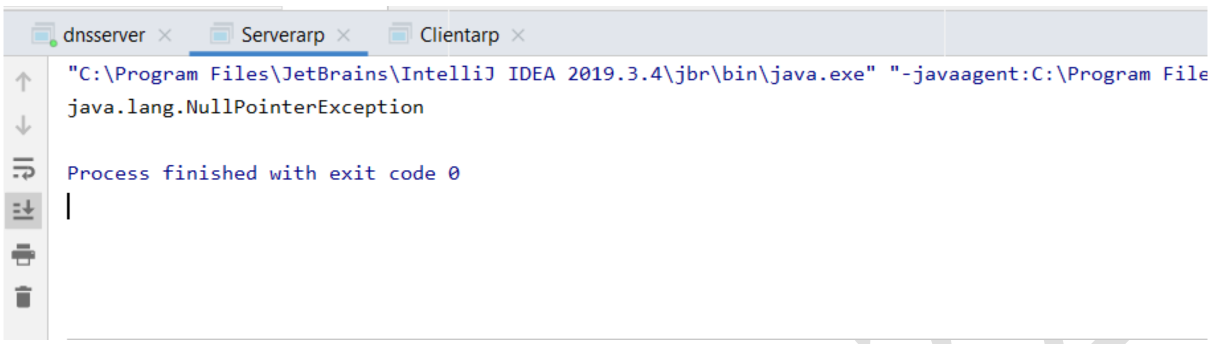
```
package program3;
import java.io.*;
import java.net.*;
import java.util.*;

public class Serverarp {
    public static void main(String args[]) {
        try {
            ServerSocket obj = new ServerSocket(5604);
            Socket obj1 = obj.accept();
            while (true) {
                {
                    DataInputStream din = new DataInputStream(obj1.getInputStream());
                    DataOutputStream dout = new
DataOutputStream(obj1.getOutputStream());
                    String str = din.readLine();
                    String ip[] = {"165.165.80.80", "165.165.79.1"};
                    String mac[] = {"8A:BC:E3:FA:92:D5", "D8-D3-85-EB-12-E3"};
                    for (int i = 0; i < ip.length; i++) {
                        if (str.equals(ip[i])) {
                            dout.writeBytes(mac[i] + '\n');
                            break;
                        }
                    }
                    obj.close();
                }
            }
        } catch (Exception e) {
            System.out.println(e);

        }
    }


}
```

client arp:

```
package program3;
import java.io.*;
import java.net.*;
import java.util.*;
public class Clientarp {
   public static void main(String args[]) {
      try {
         BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
         Socket clsct=new Socket("127.0.0.1",5604);
         DataInputStream din=new DataInputStream(clsct.getInputStream());
         DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
         System.out.println("Enter the Logical address(IP):");
         String str1=in.readLine();
         dout.writeBytes(str1+'\n');
         String str=din.readLine();
         System.out.println("The Physical Address is: "+str);
         clsct.close(); }
      catch (Exception e)
      { System.out.println(e); }
   }
}
```

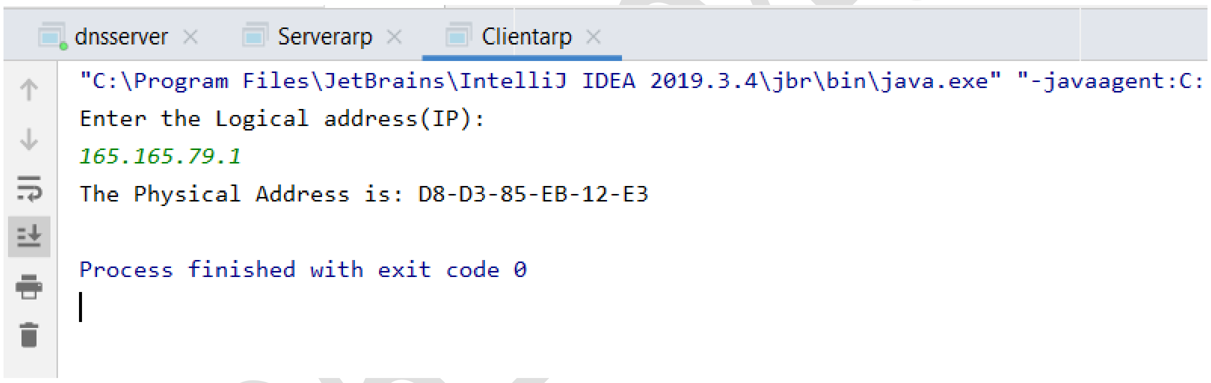
output:

server:



```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.4\jbr\bin\java.exe" "-javaagent:C:\Program File
java.lang.NullPointerException

Process finished with exit code 0
```

client:



```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.4\jbr\bin\java.exe" "-javaagent:C:
Enter the Logical address(IP):
165.165.79.1
The Physical Address is: D8-D3-85-EB-12-E3

Process finished with exit code 0
```

RARP:

ALGORITHM:

SERVER:

STEP 1: Include header files.

STEP 2: Create the socket address structure.

STEP 3: IP address, port number and family is initialized with server’s socket address structure

STEP 4: Socket address is manipulated using the byte manipulation function bzero().

STEP 5: Socket created using socket() function.

STEP 6: Using bind () function, socket is bound with the server’s well known port.

STEP 7: Enter number of client, accept those connection with those client with specified port.

STEP 8: The MAC is received from the client and the corresponding IP address is sent to the client.

STEP 9: Socket is closed.

CLIENT:

STEP 1: Include header files.

STEP 2: Socket address structure is created.

STEP 3: Server’s socket address structure is initialized with IP address, port number and family.

STEP 4: Server and the client’s address are manipulated using the byte manipulation functionbzero().

STEP 5: Socket is created using socket () function.

STEP 6: Socket is bound with the server’s well known port.

STEP 7: The MAC address is sent to the server.

STEP 9: Corresponding IP address is received from the server and it is printed.

STEP 10: Socket is closed.

Program:

Server rarp:

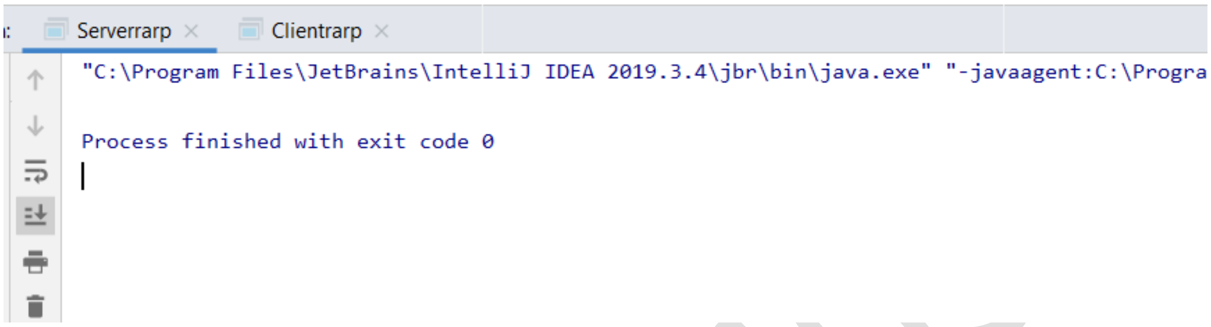
```
package program3;
import java.io.*; import java.net.*; import java.util.*;
public class Serverrarp {
   public static void main(String args[]) {
       try {
          DatagramSocket server=new DatagramSocket(1309);
          while(true)
          {
             byte[] sendbyte=new byte[1024];
             byte[] receivebyte=new byte[1024];
             DatagramPacket receiver=new
DatagramPacket(receivebyte,receivebyte.length);
             server.receive(receiver);
             String str=new String(receiver.getData());
             String s=str.trim();
             InetAddress addr=receiver.getAddress();
             int port=receiver.getPort();
             String ip[]={"254.250.146.213","254.235.18.227"};
             String mac[]={"8A:BC:E3:FA:92:D5","D8-D3-85-EB-12-E3"};
             for(int i=0;i<ip.length;i++)
             {
                if(s.equals(mac[i]))
                {
                   sendbyte=ip[i].getBytes();
                   DatagramPacket sender=new
DatagramPacket(sendbyte,sendbyte.length,addr,port);
                   server.send(sender); break;
                }
             }
             break;
          }
       }
       catch(Exception e)
       { System.out.println(e); }
   }
}
```

client rarp:

```
package program3;
import java.io.*; import java.net.*; import java.util.*;

public class Clientrarp {
   public static void main(String args[]) {
      try
      {
         DatagramSocket client=new DatagramSocket();
         InetAddress addr=InetAddress.getByName("127.0.0.1");
         byte[] sendbyte=new byte[1024];
         byte[] receivebyte=new byte[1024];
         BufferedReader in=new BufferedReader(new
InputStreamReader(System.in));
         System.out.println("Enter the Physical address (MAC):");
         String str=in.readLine();
         sendbyte=str.getBytes();
         DatagramPacket sender=new
DatagramPacket(sendbyte,sendbyte.length,addr,1309);
         client.send(sender);
         DatagramPacket receiver=new
DatagramPacket(receivebyte,receivebyte.length);
         client.receive(receiver);
         String s=new String(receiver.getData());
         System.out.println("The Logical Address is(IP): "+s.trim()); client.close();
      }
      catch(Exception e)
      { System.out.println(e); }
   }
}
```

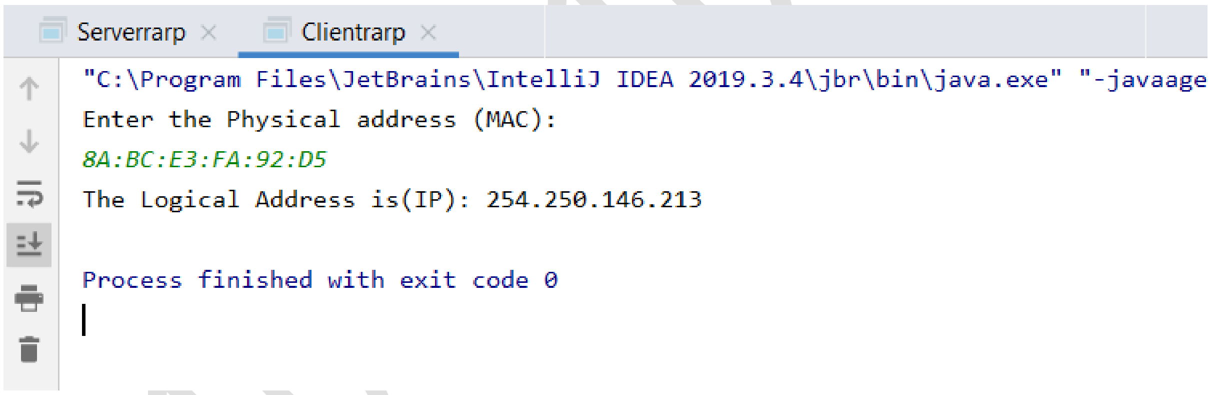
output:

server:



```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.4\jbr\bin\java.exe" "-javaagent:C:\Progra

Process finished with exit code 0
```

client:



```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.4\jbr\bin\java.exe" "-javaage
Enter the Physical address (MAC):
8A:BC:E3:FA:92:D5
The Logical Address is(IP): 254.250.146.213

Process finished with exit code 0
```

Result:

Thus the program to implement Address Resolution Protocol and Reverse Address Resolution Protocol has been written and executed successfully

EX.NO:6B SIMULATION OF CONGESTION CONTROL ALGORITHMS USING NS2

AIM:

To simulate the performance of Stop wait protocol and Go Back N Protocol using ns2.

ALGORITHM:

Step 1: Start the program

Step 2: Create the trace file and NAM file.

Step 3: Setup the topology object

Step 4: Create nodes and set duplex links between the nodes.

Step 5: Create nodes and attach them to the queue, DROPTAIL

Step 6: Configure the nodes and provides initial location of nodes. Generation of movements is done.

Step 7: Setup a TCP Connection between nodes

Step 8: Define a initialize positions for the nam window.

Step 9: Telling nodes when the simulation ends

Step 10: Ending nam and the simulation

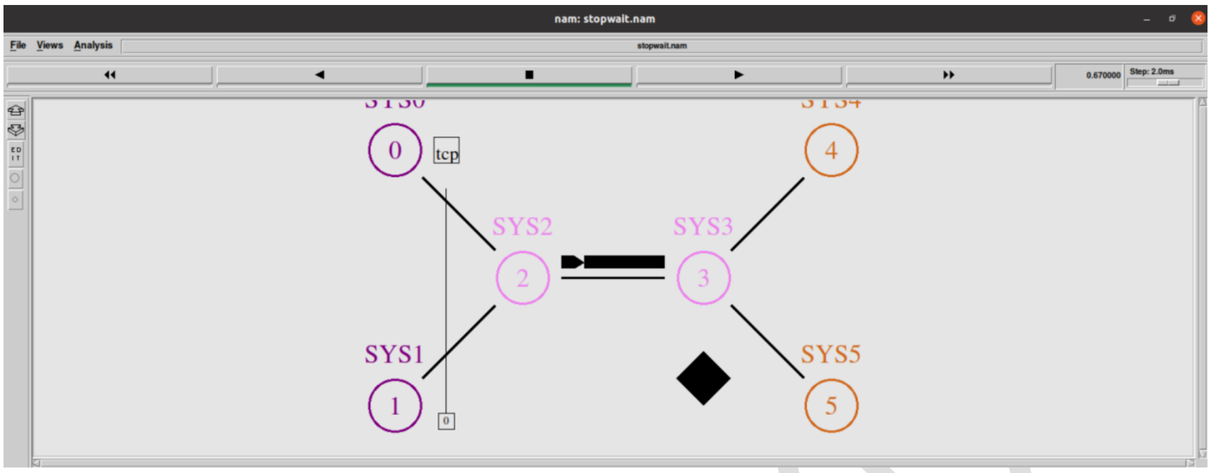
PROGRAM:

```
// stop and wait protocol
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$n0 color "purple"
$n1 color "purple"
$n2 color "violet"
$n3 color "violet"
$n4 color "chocolate"
$n5 color "chocolate"

set f [open stopwait.tr w]
$ns trace-all $f
set nf [open stopwait.nam w]
$ns namtrace-all $nf

$ns at 0.0 "$n0 label SYS0"
$ns at 0.0 "$n1 label SYS1"
$ns at 0.0 "$n2 label SYS2"
$ns at 0.0 "$n3 label SYS3"
$ns at 0.0 "$n4 label SYS4"
$ns at 0.0 "$n5 label SYS5"

$ns duplex-link $n0 $n2 0.2Mb 20ms DropTail
```

```
$ns duplex-link $n1 $n2 0.2Mb 20ms DropTail
$ns duplex-link $n2 $n3 0.2Mb 20ms DropTail
$ns duplex-link $n3 $n4 0.2Mb 20ms DropTail
$ns duplex-link $n3 $n5 0.2Mb 20ms DropTail

$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
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down
$ns queue-limit $n0 $n2 10

Agent/TCP set_nam_tracevar_true
set tcp [new Agent/TCP]
$tcp set window 1
$tcp set maxcwnd 1
$tcp set fid 1
$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns add-agent-trace $tcp tcp
$ns monitor-agent-trace $tcp
$tcp tracevar cwnd

$ns at 0.1 "$ftp start"
$ns at 0.53 "$ns queue-limit $n3 $n5 0"
```

```
$ns at 0.80 "$ns queue-limit $n3 $n5 5"
$ns at 2.0 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n5 $sink"
$ns at 2.5 "finish"

#$ns at 0.0 "$ns trace-annotate \"STOPWAIT\""
$ns at 0.01 "$ns trace-annotate \"FTP starts at 0.01\""
$ns at 0.10 "$ns trace-annotate \"Send Request SYS0 to SYS5\""
$ns at 0.18 "$ns trace-annotate \"Receive Request SYS5 to SYS0\""
$ns at 0.24 "$ns trace-annotate \"Send Packet_0 SYS0 to SYS5\""
$ns at 0.42 "$ns trace-annotate \"Receive Ack_0\""
$ns at 0.48 "$ns trace-annotate \"Send Packet_1\""
$ns at 0.60 "$ns trace-annotate \"Disconnect N2 So loss the packet1\""
$ns at 0.67 "$ns trace-annotate \"Waiting for Ack_1\""
$ns at 0.95 "$ns trace-annotate \"Send Packet_1 again\""
$ns at 1.95 "$ns trace-annotate \"Deattach SYS3,Packet_1 again\""
$ns at 2.09 "$ns trace-annotate \"Receive Ack_1\""
$ns at 2.10 "$ns trace-annotate \"SEnd Packet_2\""
$ns at 2.38 "$ns trace-annotate \"Receive Ack_2\""
$ns at 2.5 "$ns trace-annotate \"FTP stops\""

proc finish {} {
        global ns nf
        $ns flush-trace
        close $nf
        exec nam stopwait.nam &
        exit 0
}
$ns run
```

OUTPUT: stop wait



Program:

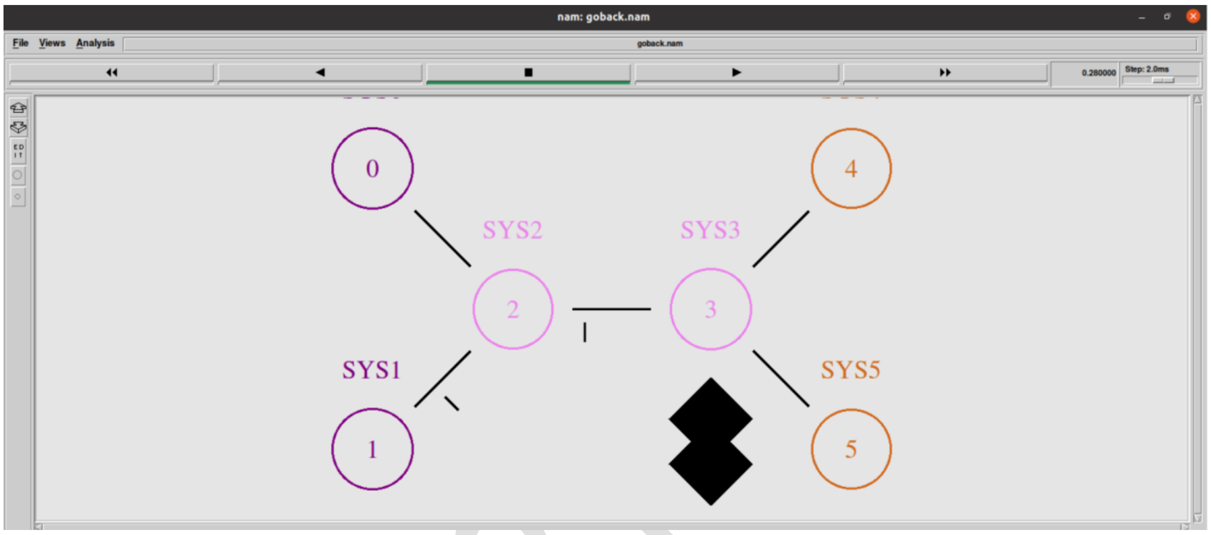
//Go Back

```
set ns [new Simulator]
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$n0 color "purple"
$n1 color "purple"
$n2 color "violet"
$n3 color "violet"
$n4 color "chocolate"
```

```
$n5 color "chocolate"

set f [open goback.tr w]
$ns trace-all $f
set nf [open goback.nam w]
$ns namtrace-all $nf

$ns at 0.0 "$n0 label SYS0"
$ns at 0.0 "$n1 label SYS1"
$ns at 0.0 "$n2 label SYS2"
$ns at 0.0 "$n3 label SYS3"
$ns at 0.0 "$n4 label SYS4"
$ns at 0.0 "$n5 label SYS5"

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$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 $n3 $n5 1Mb 10ms DropTail

$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
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n3 $n5 orient right-down

$ns queue-limit $n0 $n2 5
#Agent/TCP set nam_tracevar_true

set tcp [new Agent/TCP]
```

```
$tcp set fid 1
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.05 "$ftp start"
$ns at 0.06 "$tcp set windowlnit 6"
$ns at 0.06 "$tcp set maxcwnd 6"
$ns at 0.25 "$ns queue-limit $n3 $n4 0"
$ns at 0.26 "$ns queue-limit $n3 $n4 10"
$ns at 0.305 "$tcp set windowlnit 4"
$ns at 0.305 "$tcp set maxcwnd 4"
$ns at 0.368 "$ns detach-agent $n1 $tcp ; $ns detach-agent $n4 $sink"
$ns at 1.5 "finish"

$ns at 0.0 "$ns trace-annotate \"Goback N end\""
$ns at 0.05 "$ns trace-annotate \"FTP starts at 0.01\""
$ns at 0.06 "$ns trace-annotate \"Send 6Packets from SYS1 to SYS4\""
$ns at 0.26 "$ns trace-annotate \"Error Occurs for 4th packet so dont send ack for
the Packet\""
$ns at 0.30 "$ns trace-annotate \"Retransmit Packet_4 to 6\""
$ns at 1.0 "$ns trace-annotate \"FTP stops\""
proc finish {} {
global ns nf
$ns flush-trace
close $nf
puts "filtering..."
exec nam goback.nam &
exit 0
}
$ns run
```

OUTPUT: go back



RESULT: Thus tcl programs were executed to simulate the performance of stop wait and Go back N protocol using ns2

EX.NO: 7 Study of TCP/UDP performance using Simulation tool

Aim:

To study about TCP/UDP performance using Simulation tool

Algorithm: UDP

Step 1: Start the program

Step 2: set up a new simulator

Step 3: Create the trace file and NAM file.

Step 4: Create nodes and set all the nodes as n0,n1,n2,n3

Step 5: : Create nodes and set duplex links between the nodes.

Step 6: Create nodes and attach them to the queue, DROPTAIL,

Step 7: Configure the nodes and provides initial location of nodes. Generation of movements is done.

Step 8: Setup a UDP Connection between nodes

Step 9: Define a initialize positions for the nam window.

Step 10: Telling nodes when the simulation ends

Step 11: Ending nam and the simulation

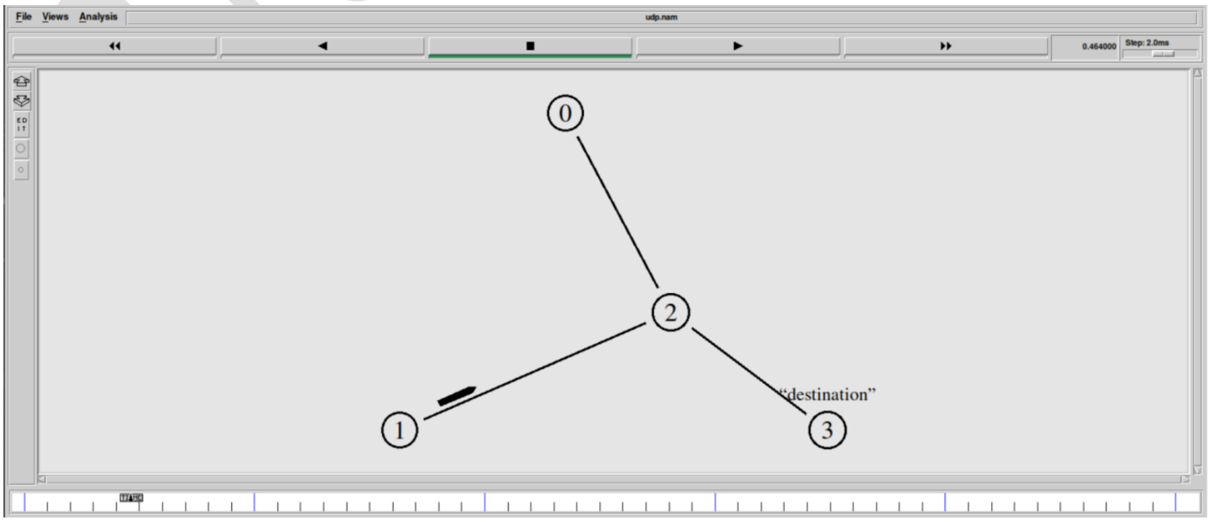
Program:

UDP:

```
set ns [new Simulator]
set nf [open udp.nam w]
$ns namtrace-all $nf

set tf [open mo.tr w]
$ns trace-all $tf

proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam udp.nam &
exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]


$n3 label “destination”

$ns duplex-link $n0 $n2 10Mb 1ms DropTail
$ns duplex-link $n1 $n2 10Mb 1ms DropTail
$ns duplex-link $n2 $n3 10Mb 1ms DropTail

$ns queue-limit $n0 $n2 10
$ns queue-limit $n1 $n2 10

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0

set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
```

```
set cbr1 [new Application/Traffic/CBR]
$cbr1 attach-agent $udp1

set udp2 [new Agent/UDP]
$ns attach-agent $n2 $udp2

set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2

set null0 [new Agent/Null]
$ns attach-agent $n3 $null0

$ns connect $udp0 $null0
$ns connect $udp1 $null0
$ns connect $udp2 $null0

$ns at 0.1 "$cbr1 start"
$ns at 0.2 "$cbr0 start"
$ns at 5 "finish"
$ns run
```

Output:UDP



Algorithm: TCP:

Step 1: Start the program

Step 2: set up a new simulator

Step 3: Create the trace file and NAM file.

Step 4: Create nodes and set all the nodes as n0,n1,n2,n3

Step 5: : Create nodes and set duplex links between the nodes.

Step 6: Create nodes and attach them to the queue, DROPTAIL,

Step 7: Configure the nodes and provides initial location of nodes. Generation of movements is done.

Step 8: Setup a TCP Connection between nodes

Step 9: Define a initialize positions for the nam window.

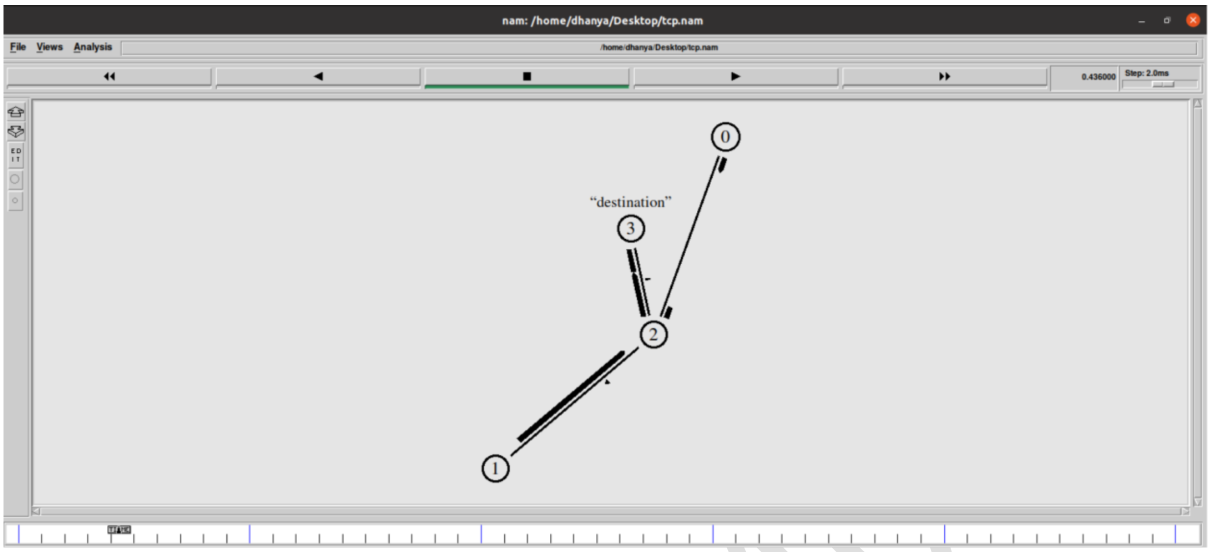
Step 10: Telling nodes when the simulation ends

Step 11: Ending nam and the simulation

Program:

```
set ns [new Simulator]
set nf [open tcp.nam w]
$ns namtrace-all $nf

set tf [open tcp.tr w]
$ns trace-all $tf

proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam tcp.nam &
exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$n3 label “destination”

$ns duplex-link $n0 $n2 10Mb 1ms DropTail
$ns duplex-link $n1 $n2 10Mb 1ms DropTail
$ns duplex-link $n2 $n3 10Mb 1ms DropTail


set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0

set ftp0 [new Application/FTP]
$ftp0 set packet_Size_ 500
$ftp0 set interval_ 0.005
$ftp0 attach-agent $tcp0

set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1

set ftp1 [new Application/FTP]
$ftp1 set packet_Size_ 500
$ftp1 set interval_ 0.005
$ftp1 attach-agent $tcp1

set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0

set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1

$ns connect $tcp0 $sink0
$ns connect $tcp1 $sink1


$ns at 0.1 "$ftp1 start"
$ns at 0.1 "$ftp0 start"
$ns at 5 "finish"
$ns run
```

Output:


nam: /home/dhanya/Desktop/tcp.nam
File
Views
Analysis
/home/dhanya/Desktop/tcp.nam
0.436000
Step: 2.0ms
"destination"
0
1
2
3

Result: Thus the TCP/UDP performance using Simulation tool was studied and implemented.

EX NO:8 Simulation of Distance Vector/ Link State Routing algorithm.

AIM:

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

DESCRIPTION:

Distance Vector Routing is one of the routing algorithm in a Wide Area Network for computing shortest path between source and destination. The Router is one main devices used in a wide area network. The main task of the router is Routing. It forms the routing table and delivers the packets depending upon the routes in the table-either directly or via an intermediate devices. Each router initially has information about its all neighbors. Then this information will be shared among nodes.

ALGORITHM:

1. Define new simulator
2. Define different colors for data flows (for NAM)
3. Define a new Trace file and open it
4. Define a new NAM Trace file and open it
5. Define a 'finish' procedure – to flush trace record in the `trace and trace output files.
6. Define the routing protocol as Distance Vector (DV)
7. Create six nodes – n0,n1,..n5
8. Create links between the nodes with 0.3Mb and 10 ms Link with DropTail option
9. Give node position (for NAM) to place six nodes in the layout
10. Setup a TCP connection – attach TCP Source Agent to node n0 and TCP sink agent to node n5
11. Setup a FTP over TCP connection
12. Define configuration such that link between nodes n1 and n4 to be failed at 1.0 interval, and up again at 4.5 interval
13. Start the simulation.

PROGRAM

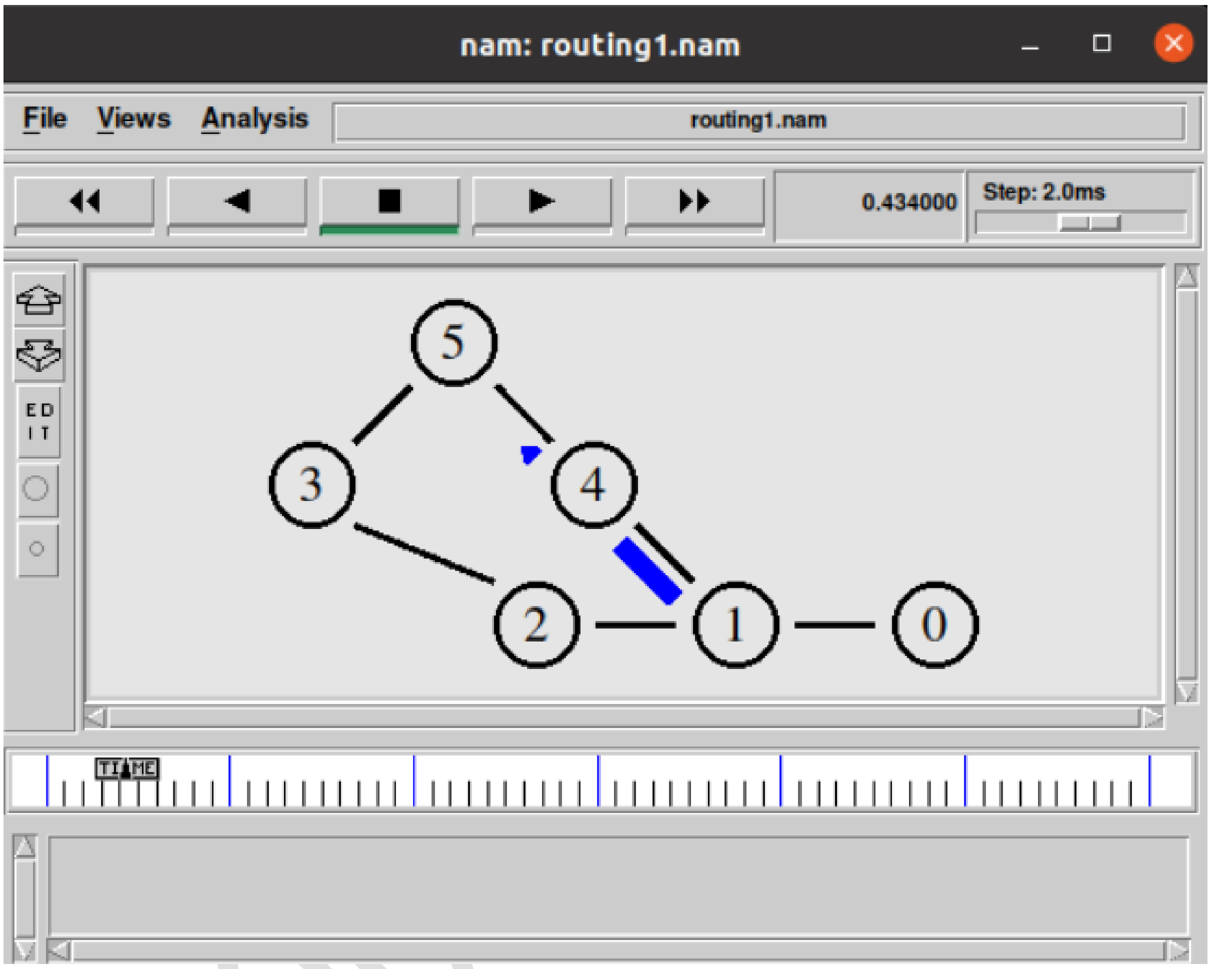
routing1.tcl

```
set ns [new Simulator]

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

#Open the Trace file
set file1 [open routing1.tr w]
$ns trace-all $file1

#Open the NAM trace file
set file2 [open routing1.nam w]
$ns namtrace-all $file2

#Define a 'finish' procedure
proc finish {} {
        global ns file1 file2
        $ns flush-trace
        close $file1
        close $file2
        exec nam routing1.nam &
        exit 0
}

# Next line should be commented out to have the static routing
```

```
$ns rtproto DV

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

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

#Give node position (for NAM)

$ns duplex-link-op $n0 $n1 orient left
$ns duplex-link-op $n1 $n2 orient left
$ns duplex-link-op $n2 $n3 orient left-up
$ns duplex-link-op $n1 $n4 orient left-up
$ns duplex-link-op $n4 $n5 orient left-up
$ns duplex-link-op $n3 $n5 orient right-up
#Setup a TCP connection
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
```

```
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

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

$ns rtmodel-at 1.0 down $n1 $n4
$ns rtmodel-at 3.0 up $n1 $n4

$ns at 0.1 "$ftp start"

$ns at 6.0 "finish"

$ns run
```

Output:


nam: routing1.nam
File
Views
Analysis
routing1.nam
0.434000
Step: 2.0ms
EDIT
TIME
5
3
4
2
1
0

EX NO: 8 B Simulation of Link State Routing algorithm

AIM:

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

ALGORITHM:

1. Define new simualtor
2. Define different colors for data flows (for NAM)
3. Define a new Trace file and open it
4. Define a new NAM Trace file and open it
5. Define a 'finish' procedure – to flush trace record in the `trace and trace output files.
6. Define the routing protocol as Link State (LS)
7. Create six nodes – n0,n1,..n5
8. Create links between the nodes with 0.3Mb and 10 ms Link with DropTail option
9. Give node position (for NAM) to place six nodes in the layout
10. Setup a TCP connection – attach TCP Source Agent to node n0 and TCP sink agent to node n5
11. Setup a FTP over TCP connection
12. Define configuration such that link between nodes n1 and n4 to be failed at 1.0 interval, and up again at 4.5 interval
13. Start the simulation

Program:

// Link state routing

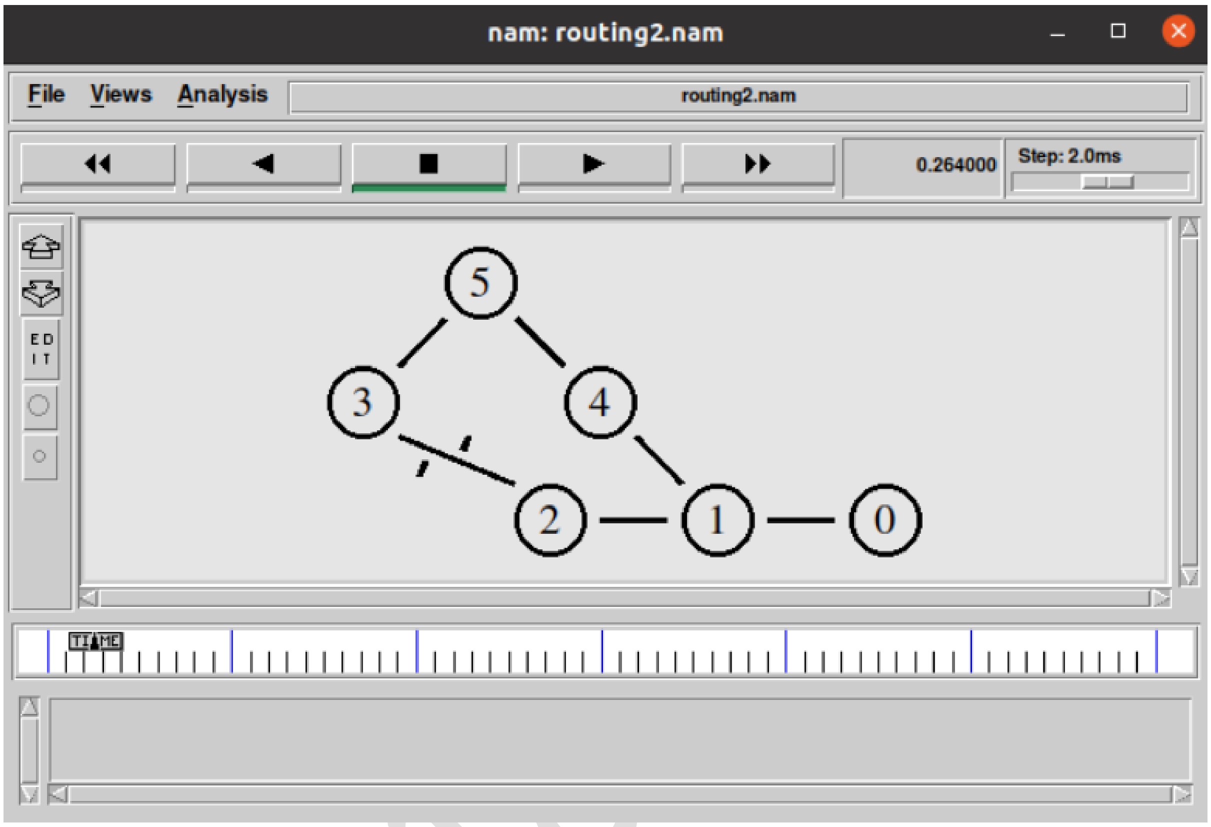
#routing2.tcl

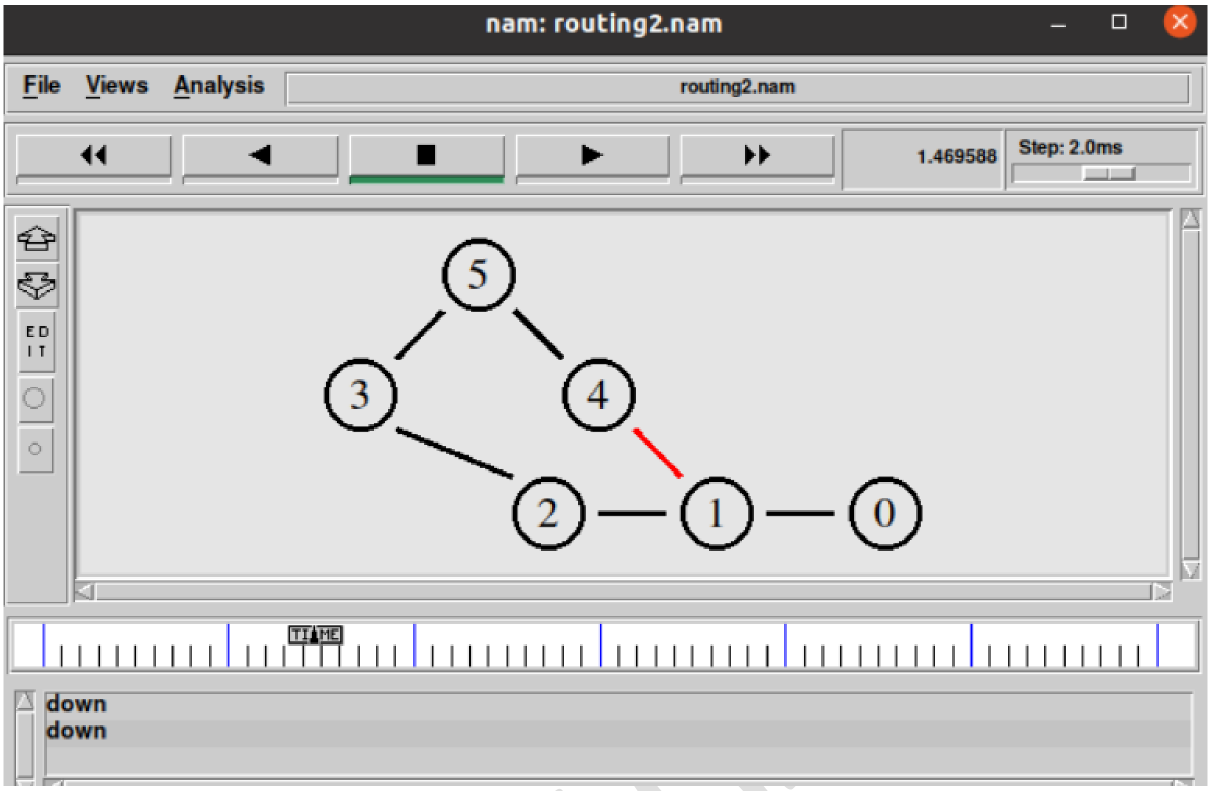
```
set ns [new Simulator]

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

#Open the Trace file
set file1 [open routing2.tr w]
$ns trace-all $file1

#Open the NAM trace file
set file2 [open routing2.nam w]
$ns namtrace-all $file2

#Define a 'finish' procedure
proc finish {} {
        global ns file1 file2
        $ns flush-trace
        close $file1
        close $file2
        exec nam routing2.nam &
        exit 0
}

# Next line should be commented out to have the static routing
```

```
$ns rtproto LS

#Create six 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]

#Create links between the nodes
$ns duplex-link $n0 $n1 0.3Mb 10ms DropTail
$ns duplex-link $n1 $n2 0.3Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.3Mb 10ms DropTail
$ns duplex-link $n1 $n4 0.3Mb 10ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 10ms DropTail
$ns duplex-link $n4 $n5 0.5Mb 10ms DropTail
#Give node position (for NAM)
$ns duplex-link-op  $n0 $n1 orient right
$ns duplex-link-op $n0 $n1 orient left
$ns duplex-link-op $n1 $n2 orient left
$ns duplex-link-op $n2 $n3 orient left-up
$ns duplex-link-op $n1 $n4 orient left-up
$ns duplex-link-op $n4 $n5 orient left-up
$ns duplex-link-op $n3 $n5 orient right-up

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

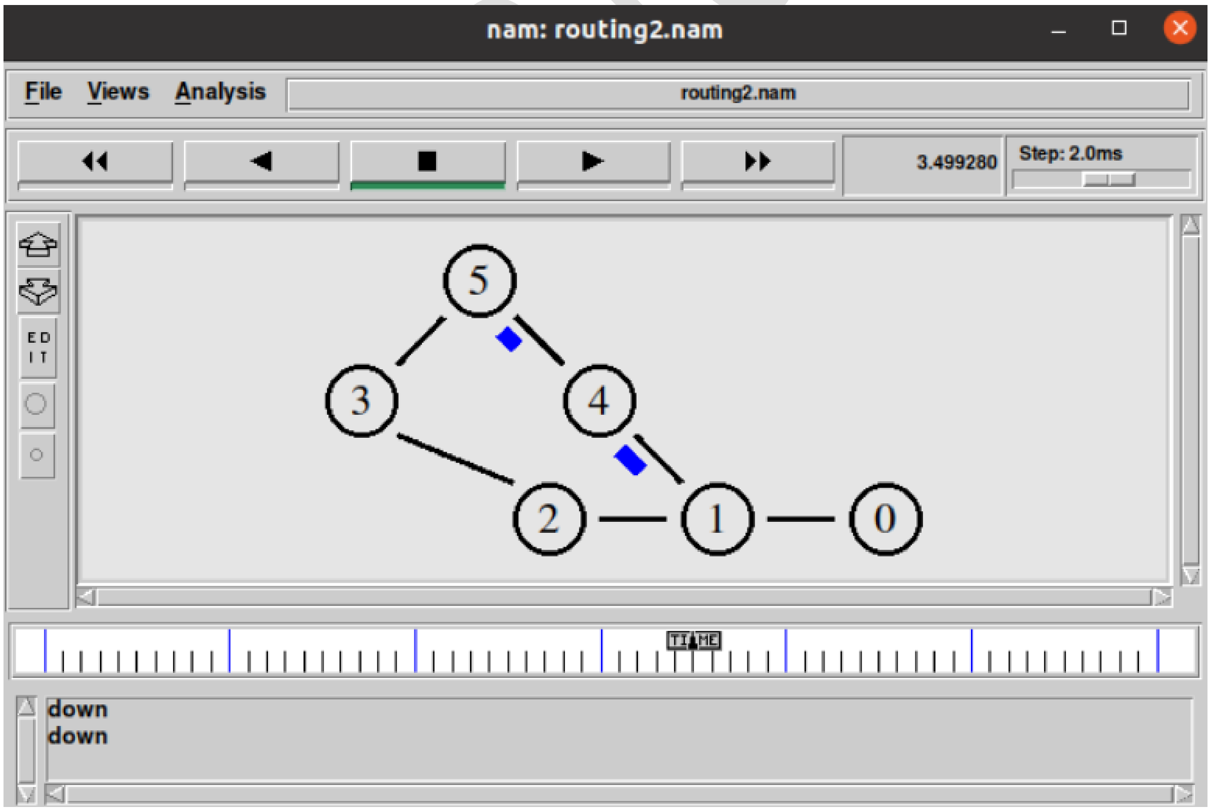
```
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

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

$ns rtmodel-at 1.0 down $n1 $n4
$ns rtmodel-at 3.0 up $n1 $n4

$ns at 0.1 "$ftp start"

$ns at 6.0 "finish"

$ns run
```

Output:


nam: routing2.nam
File
Views
Analysis
routing2.nam
0.264000
Step: 2.0ms
ED IT
TIME
5
3
4
2
1
0

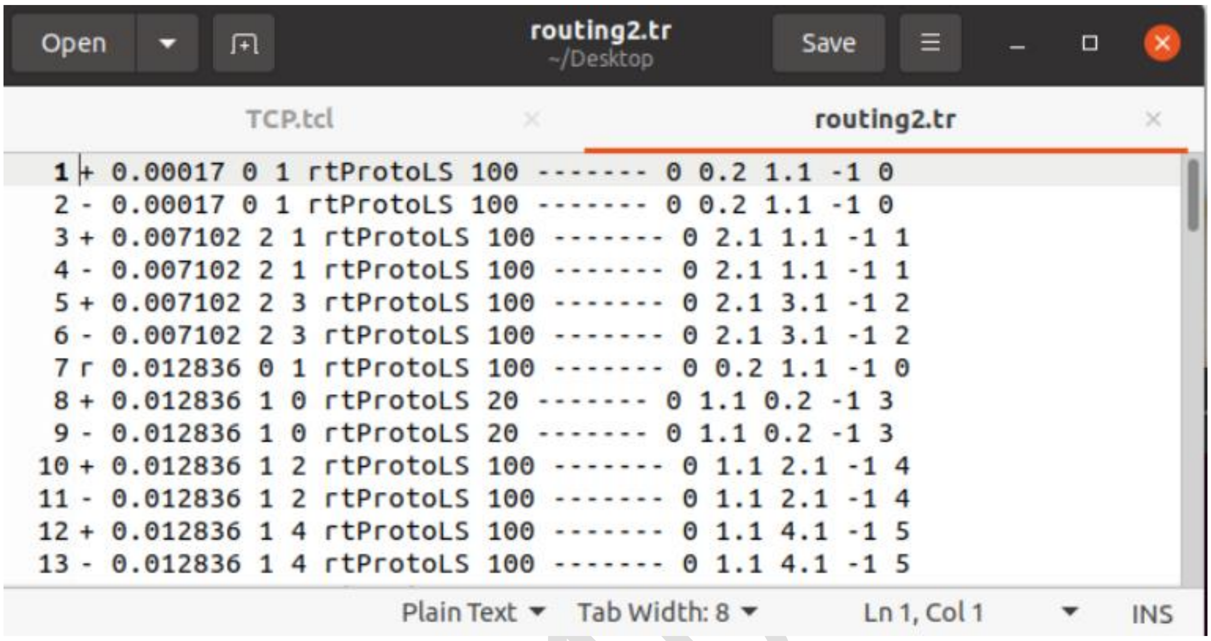


down
down



down
down

Trace route:



```
+ 0.00017 0 1 rtProtoLS 100 ------- 0 0.2 1.1 -1 0
- 0.00017 0 1 rtProtoLS 100 ------- 0 0.2 1.1 -1 0
+ 0.007102 2 1 rtProtoLS 100 ------- 0 2.1 1.1 -1 1
- 0.007102 2 1 rtProtoLS 100 ------- 0 2.1 1.1 -1 1
+ 0.007102 2 3 rtProtoLS 100 ------- 0 2.1 3.1 -1 2
- 0.007102 2 3 rtProtoLS 100 ------- 0 2.1 3.1 -1 2
r 0.012836 0 1 rtProtoLS 100 ------- 0 0.2 1.1 -1 0
+ 0.012836 1 0 rtProtoLS 20 ------- 0 1.1 0.2 -1 3
- 0.012836 1 0 rtProtoLS 20 ------- 0 1.1 0.2 -1 3
+ 0.012836 1 2 rtProtoLS 100 ------- 0 1.1 2.1 -1 4
- 0.012836 1 2 rtProtoLS 100 ------- 0 1.1 2.1 -1 4
+ 0.012836 1 4 rtProtoLS 100 ------- 0 1.1 4.1 -1 5
- 0.012836 1 4 rtProtoLS 100 ------- 0 1.1 4.1 -1 5
```

Result: Thus the simulation of link state routing algorithm was implemented and output was verified.

EX NO: 9 Performance evaluation of Routing protocols using Simulation tool.

AIM:

To implement different routing algorithms and compare their performance using network simulator (ns2)

A) LINK STATE ROUTING ALGORITHM

ALGORITHM:

1. Define new simualtor
2. Define different colors for data flows (for NAM)
3. Define a new Trace file and open it
4. Define a new NAM Trace file and open it
5. Define a 'finish' procedure – to flush trace record in the `trace and trace output files.
6. Define the routing protocol as Link State (LS)
7. Create six nodes – n0, n1,..n5
8. Create links between the nodes with 0.3Mb and 10 ms Link with DropTail option
9. Give node position (for NAM) to place six nodes in the layout
10. Setup a TCP connection – attach TCP Source Agent to node n0 and TCP sink agent to node n5
11. Setup a FTP over TCP connection
12. Define configuration such that link between nodes n1 and n4 to be failed at 1.0 interval, and up again at 4.5 interval
13. Start the simulation

Program:

//Link state routing

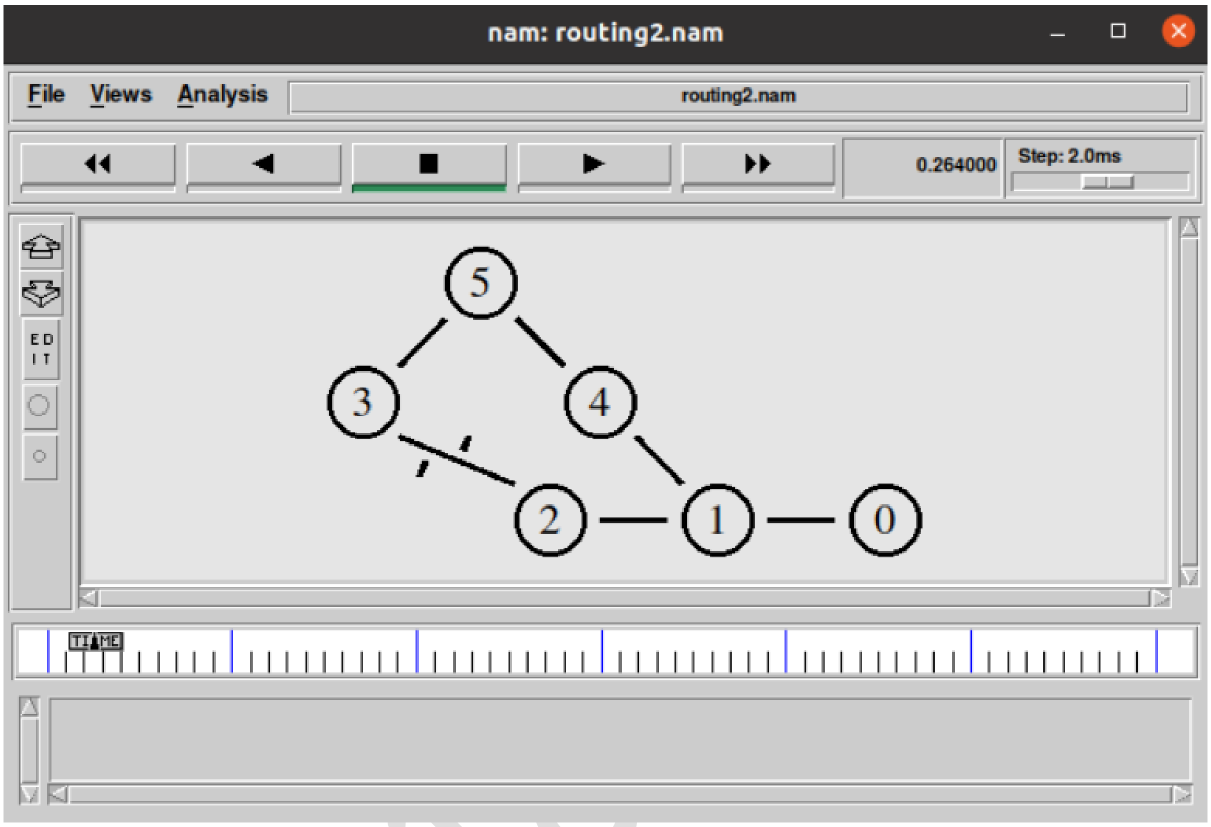
#routing2.tcl

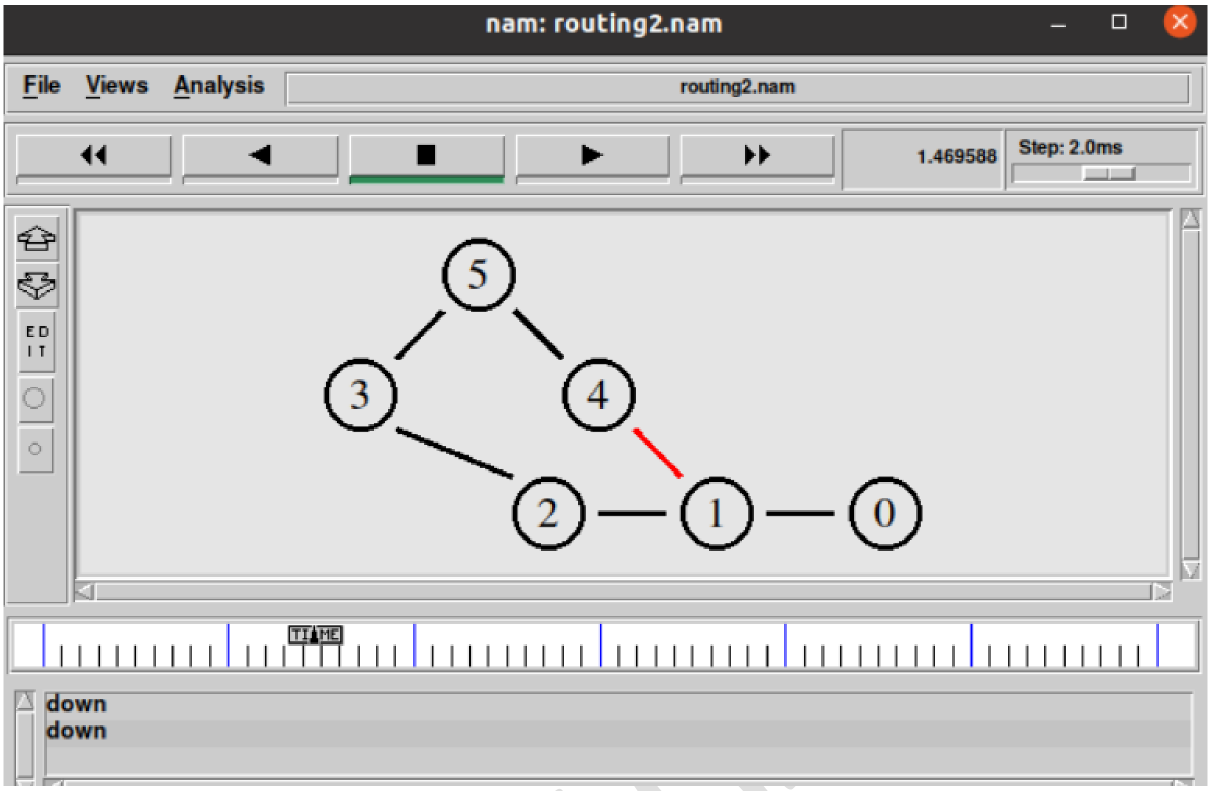
```
set ns [new Simulator]

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

#Open the Trace file
set file1 [open routing2.tr w]
$ns trace-all $file1

#Open the NAM trace file
set file2 [open routing2.nam w]
$ns namtrace-all $file2

#Define a 'finish' procedure
proc finish {} {
        global ns file1 file2
        $ns flush-trace
        close $file1
        close $file2
        exec nam routing2.nam &
        exit 0
}
```

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

#Create six 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]

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

#Give node position (for NAM)

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

#Setup a TCP connection
set tcp [new Agent/TCP/Newreno]
```

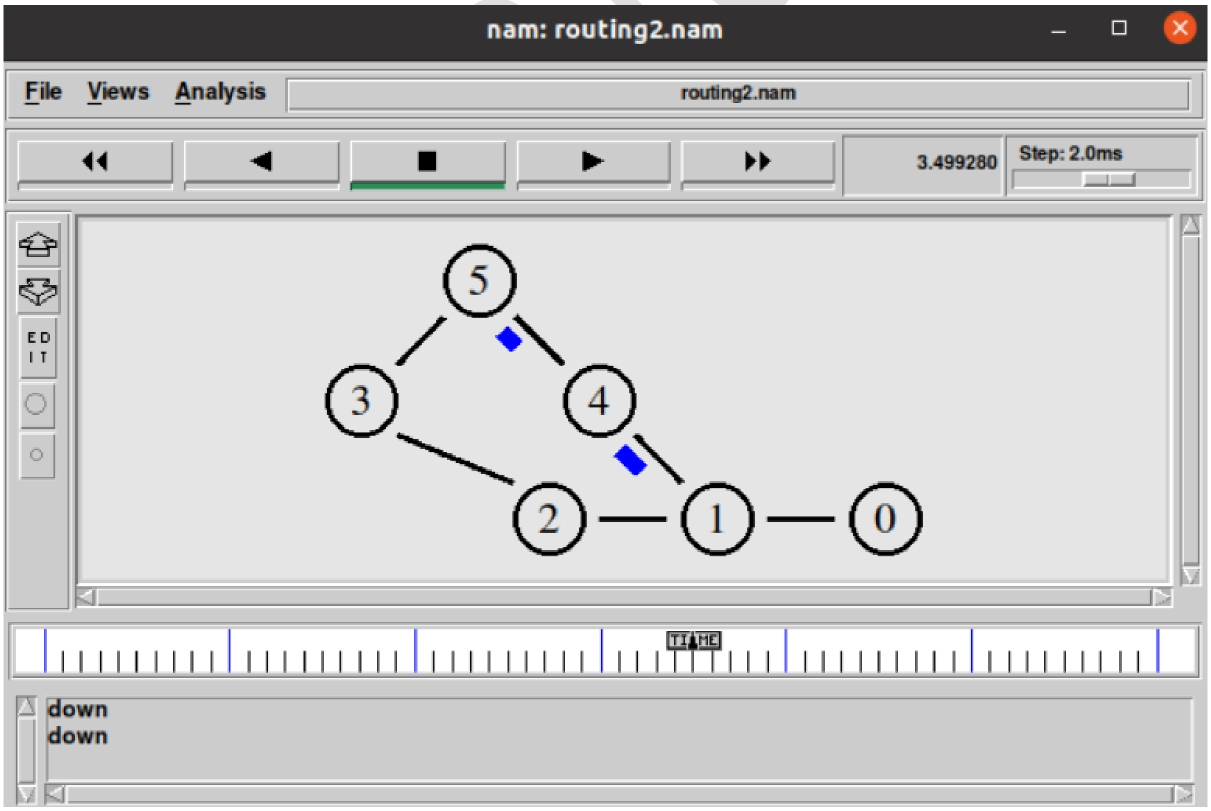
```
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

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

$ns rtmodel-at 1.0 down $n1 $n4
$ns rtmodel-at 3.0 up $n1 $n4

$ns at 0.1 "$ftp start"

$ns at 6.0 "finish"

$ns run
```

Ouput:


nam: routing2.nam
File
Views
Analysis
routing2.nam
0.264000
Step: 2.0ms
EDIT
TIME
5
3
4
2
1
0



down
down



down
down

b) FLOODING:

ALGORITHM:

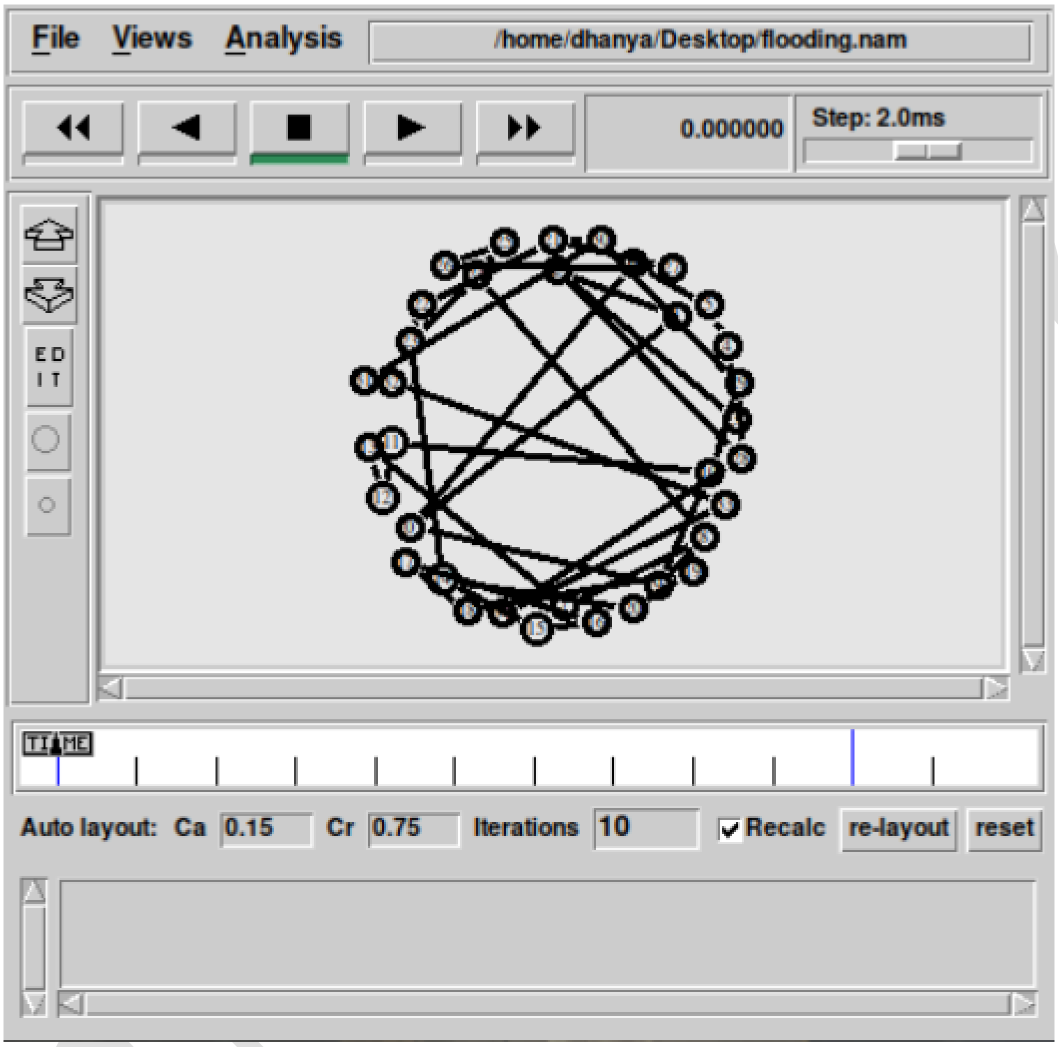
1. Define new simulator
2. Define different colors for data flows (for NAM)
3. Define a new Trace file and open it
4. Define a new NAM Trace file and open it
5. Define a 'finish' procedure – to flush trace record in the `trace and trace output files.
6. Define the flooding protocol
7. Create six nodes – n0,n1,..n5
8. Create links between the nodes with 0.3Mb and 10 ms Link with DropTail option
9. Give node position (for NAM) to place six nodes in the layout
10. Setup a TCP connection – attach TCP Source Agent to node n0 and TCP sink agent to node n5
11. Setup a FTP over TCP connection
12. Define configuration such that link between nodes n1 and n4 to be failed at 1.0 interval, and up again at 4.5 interval
13. Each node should act as both a transmitter and a receiver.
14. Each node forwards every message to every one of its neighbors except the source node.
15. Start the simulation

Program:

```
set MESSAGE_PORT 42



# parameters for topology generator
set group_size 7
set num_groups 5
set num_nodes [expr $group_size * $num_groups]



set ns [new Simulator]


set f [open flooding.tr w]
$ns trace-all $f
set nf [open flooding.nam w]
$ns namtrace-all $nf




# subclass Agent/MessagePassing to make it do flooding


Class Agent/MessagePassing/Flooding -superclass Agent/MessagePassing


Agent/MessagePassing/Flooding instproc send_message {size msgid msg} {
    $self instvar messages_seen node_
    global ns MESSAGE_PORT


    $ns trace-annotate "Node [$node_ node-addr] is sending {$msgid:$msg}"
    lappend messages_seen $msgid
    $self send_to_neighbors -1 $MESSAGE_PORT $size "$msgid:$msg"
}

Agent/MessagePassing/Flooding instproc send_to_neighbors {skip port size data} {
    $self instvar node_

        foreach x [$node_ neighbors] {
            set addr [$x set address_]
            if {$addr != $skip} {
                $self sendto $size $data $addr $port
            }
        }
}

Agent/MessagePassing/Flooding instproc recv {source sport size data} {
    $self instvar messages_seen node_
    global ns

    # extract message ID from message
    set message_id [lindex [split $data ":"] 0]

    if {[lsearch $messages_seen $message_id] == -1} {
        lappend messages_seen $message_id
      $ns trace-annotate "Node [$node_ node-addr] received {$data}"
        $self send_to_neighbors $source $sport $size $data
    } else {
        $ns trace-annotate "Node [$node_ node-addr] received redundant copy of
message #$message_id"
```

```
    }
}



## Topology Generator


# create a bunch of nodes
for {set i 0} {$i < $num_nodes} {incr i} {
    set n($i) [$ns node]
}


# create links between the nodes
for {set g 0} {$g < $num_groups} {incr g} {
    for {set i 0} {$i < $group_size} {incr i} {
        $ns duplex-link $n([expr $g*$group_size+$i]) $n([expr
$g*$group_size+($i+1)%$group_size]) 2Mb 15ms DropTail
    }
    $ns duplex-link $n([expr $g*$group_size]) $n([expr
(($g+1)%$num_groups)*$group_size+2]) 2Mb 15ms DropTail
    if {$g%2} {
        $ns duplex-link $n([expr $g*$group_size+3]) $n([expr
(($g+3)%$num_groups)*$group_size+1]) 2Mb 15ms DropTail
    }
}



# attach a new Agent/MessagePassing/Flooding to each node on port
$MESSAGE_PORT
for {set i 0} {$i < $num_nodes} {incr i} {
    set a($i) [new Agent/MessagePassing/Flooding]
    $n($i) attach  $a($i) $MESSAGE_PORT
    $a($i) set messages_seen {}
}


# now set up some events
$ns at 0.2 "$a(5) send_message 900 1 {first message}"
$ns at 0.5 "$a(17) send_message 700 2 {another one}"
$ns at 1.0 "$a(24) send_message 500 abc {yet another one}"

$ns at 2.0 "finish"

proc finish {} {
        global ns f nf
        $ns flush-trace
        close $f
        close $nf

        puts "running nam..."
        exec nam flooding.nam &
        exit 0
}

$ns run
```

Output:


File
Views
Analysis
/home/dhanya/Desktop/flooding.nam
0.000000
Step: 2.0ms
EDIT
TIME
Auto layout:
Ca
0.15
Cr
0.75
Iterations
10
Recalc
re-layout
reset

c) DISTANCE VECTOR ROUTING ALGORITHM

ALGORITHM:

1. Define new simulator
2. Define different colors for data flows (for NAM)
3. Define a new Trace file and open it
4. Define a new NAM Trace file and open it
5. Define a 'finish' procedure – to flush trace record in the `trace and trace output files.
6. Define the routing protocol as Distance Vector (DV)
7. Create six nodes – n0,n1,..n5
8. Create links between the nodes with 0.3Mb and 10 ms Link with DropTail option
9. Give node position (for NAM) to place six nodes in the layout
10. Setup a TCP connection – attach TCP Source Agent to node n0 and TCP sink agent to node n5
11. Setup a FTP over TCP connection
12. Define configuration such that link between nodes n1 and n4 to be failed at 1.0 interval, and up again at 4.5 interval
13. Start the simulation

Distance Vector Routing Algorithm

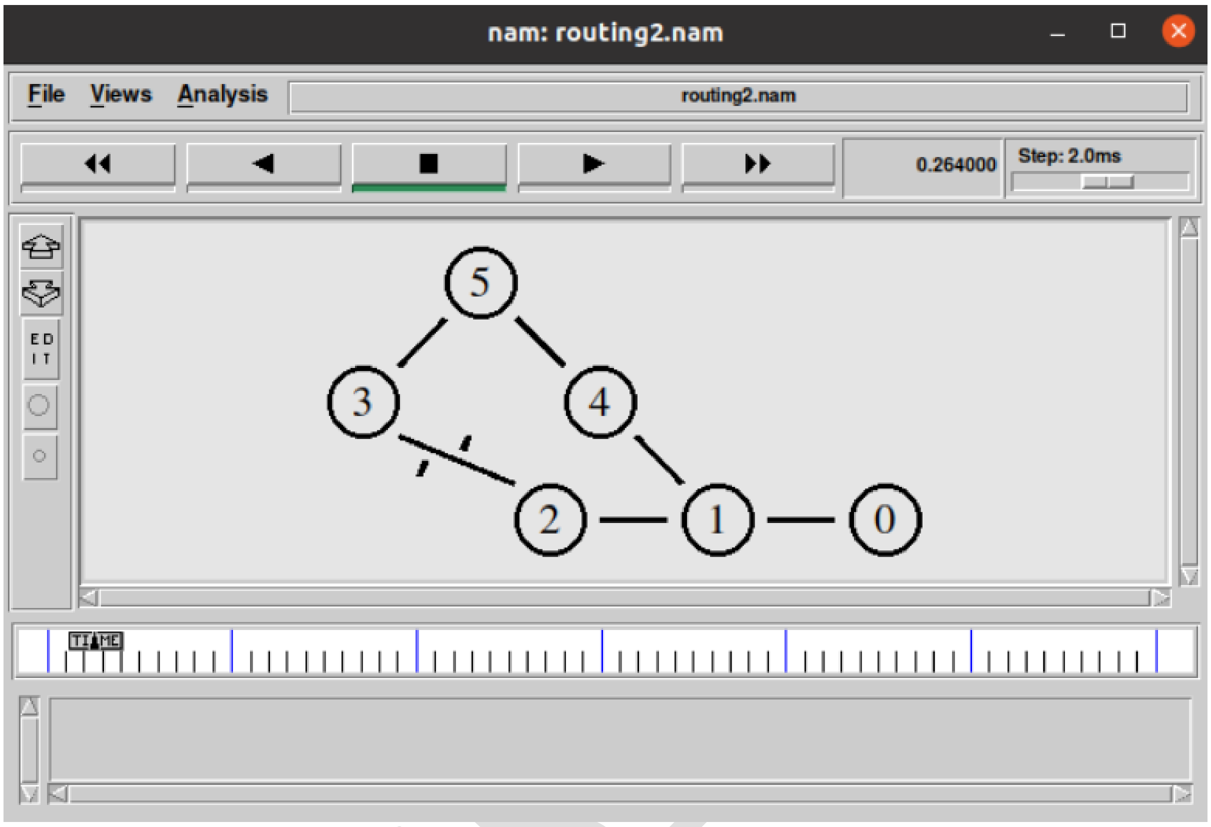
routing1.tcl

```
set ns [new Simulator]


#Define different colors for data flows (for NAM)

$ns color 1 Blue

$ns color 2 Red


#Open the Trace file

set file1 [open routing1.tr w]
```

```
$ns trace-all $file1

#Open the NAM trace file
set file2 [open routing1.nam w]
$ns namtrace-all $file2

#Define a 'finish' procedure
proc finish {} {
        global ns file1 file2
        $ns flush-trace
        close $file1
        close $file2
        exec nam routing1.nam &
        exit 0
}

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

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

#Create links between the nodes
$ns duplex-link $n0 $n1 0.3Mb 10ms DropTail
$ns duplex-link $n1 $n2 0.3Mb 10ms DropTail
```

```
$ns duplex-link $n2 $n3 0.3Mb 10ms DropTail
$ns duplex-link $n1 $n4 0.3Mb 10ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 10ms DropTail
$ns duplex-link $n4 $n5 0.5Mb 10ms DropTail

#Give node position (for NAM)

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

#Setup a TCP connection
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
$ns rtmodel-at 1.0 down $n1 $n4
$ns rtmodel-at 3.0 up $n1 $n4
$ns at 0.1 "$ftp start"
$ns at 6.0 "finish"
$ns run
```

Output:

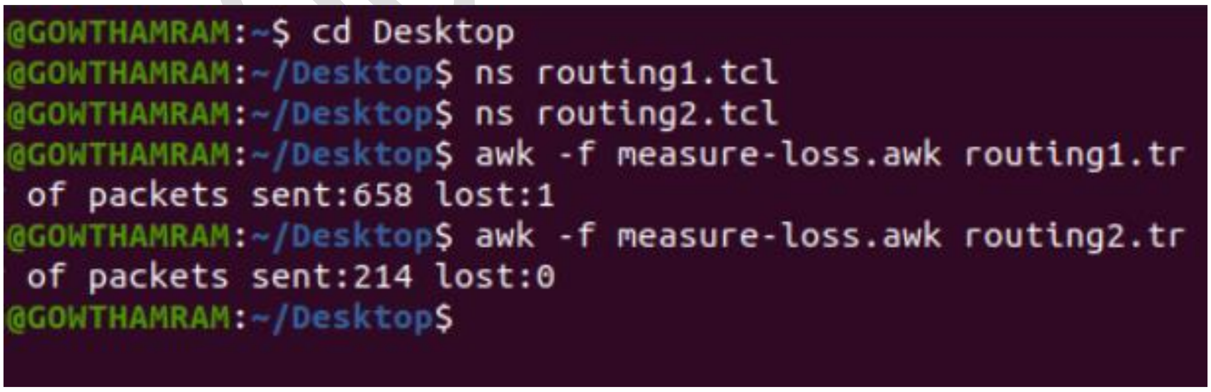


PROGRAM:

measure-loss.awk

```
BEGIN {
# Initialization. Set two variables. fsDrops: packets drop. numFs: packets sent
fsDrops = 0;
numFs = 0;
}
{
action = $1;
time = $2;
from = $3;
to = $4;
type = $5;
pktsize = $6;
flow_id = $8;
src = $9;
dst = $10;
seq_no = $11;
packet_id = $12;
if (src==0.0 && dst==5.0 && action == "+")
numFs++;
if (action == "d")
fsDrops++;
}
END {
printf("number of packets sent:%d lost:%d\n", numFs, fsDrops);
}
```

Output:



```
@GOWTHAMRAM:~$ cd Desktop
@GOWTHAMRAM:~/Desktop$ ns routing1.tcl
@GOWTHAMRAM:~/Desktop$ ns routing2.tcl
@GOWTHAMRAM:~/Desktop$ awk -f measure-loss.awk routing1.tr
 of packets sent:658 lost:1
@GOWTHAMRAM:~/Desktop$ awk -f measure-loss.awk routing2.tr
 of packets sent:214 lost:0
@GOWTHAMRAM:~/Desktop$
```

Result: thus, the performance of the routing algorithms was successfully implemented.

EX NO: 10 Simulation of error correction code (like CRC).

AIM:

To implement error detection and error correction techniques.

ALGORITHM:

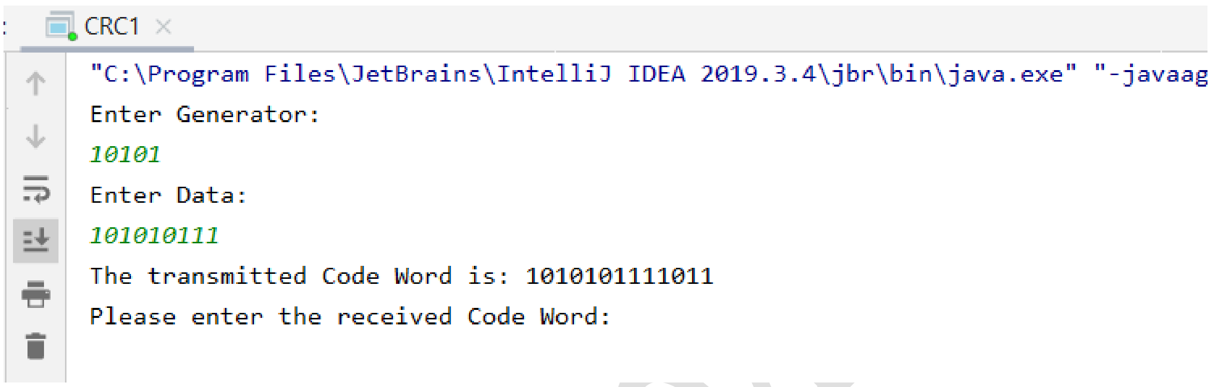
1. Start the program execution
2. Get the input in the form of bits.
3. Append 16 zeros as redundancy bits.
4. Divide the appended data using a divisor polynomial.
5. The resulting data should be transmitted to the receiver.
6. At the receiver the received data is entered.
7. The same process is repeated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

Program:

```
package program3;
import java.io.*;
public class CRC1 {
   public static void main(String args[]) throws IOException
   {
      BufferedReader br = new BufferedReader(new
InputStreamReader(System.in)); System.out.println("Enter Generator:");
      String gen = br.readLine();
      System.out.println("Enter Data:");
      String data = br.readLine();
      String code = data;
      while(code.length() < (data.length() + gen.length() - 1))
         code = code + "0";
      code = data + div(code,gen);
      System.out.println("The transmitted Code Word is: " + code);
System.out.println("Please enter the received Code Word: "); String rec =
br.readLine();
      if(Integer.parseInt(div(rec,gen)) == 0)
         System.out.println("The received code word contains no errors."); else
         System.out.println("The received code word contains errors."); }

   static String div(String num1,String num2) {
      int pointer = num2.length();
      String result = num1.substring(0, pointer); String remainder = "";
      for(int i = 0; i < num2.length(); i++)
      {
         if(result.charAt(i) == num2.charAt(i)) remainder += "0";
         else
            remainder += "1";
      }
      while(pointer < num1.length()) {
         if(remainder.charAt(0) == '0') {
            remainder = remainder.substring(1, remainder.length()); remainder =
remainder + String.valueOf(num1.charAt(pointer)); pointer++;
         }
         result = remainder;
         remainder = "";
         for(int i = 0; i < num2.length(); i++) {
            if(result.charAt(i) == num2.charAt(i)) remainder += "0";
            else
               remainder += "1";
         } }
```

```
        return remainder.substring(1,remainder.length()); }
}
```

output:



```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.3.4\jbr\bin\java.exe" "-javaag
Enter Generator:
10101
Enter Data:
101010111
The transmitted Code Word is: 1010101111011
Please enter the received Code Word:
```

Result: thus the CRC program was successfully executed

<u>CONTENT BEYOND THE SYLLABUS</u>

EX No 11 **BIT STUFFING**

<u>AIM</u>

To write a program that takes a binary file as input and performs the bit stuffing.

<u>ALGORITHM</u>

<u>Server</u>

1. Include necessary header files to support functions for Socket definitions, Socket Types, Internet

   addresses, I/Ofunctions, Unix system calls.

2. Declare variables for Socket ID, Portnumber, Socket addresses, buffer,etc.

3. Create Socket for server.

4. Bind socket with addresses.

5. Specify number of allowed connections.

6. Wait for connection.

7. Accept connection (If any).

8. Take the binary input file.

9. Perform a bit stuffing by replacing every sixth bit by 0, if it is 1 , else check the next sixth bit and repeat

   the process

10. Repeat the steps 8and 9 until the socket is closed.

11. Retrieve information from Connected Socket and display it.

12. Send information to Connected Socket.

13. Close connected socket.

<u>Client</u>

1. Include necessary header files to support functions for Socket definitions, Socket types, Internet

   addresses, I/O functions, Unix system calls.

2. Declare variables for Socket ID, Portnumber, Socket addresses, Character buffer, etc.

3. Create Socket for Client.

4. Connect client socket to the server socket addresses.

5. Enter the source file name for which the bit stuffing is performed.

6. Repeat the steps 8and 9 until the socket is closed.

7. Send information to Connected Socket

8. Retrieve information from Connected Socket and display it..

9. Close connected socket.

BIT STUFFING

SERVER

```
#include<stdio.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<netdb.h>

#include<fcntl.h>

#include<stdlib.h>

#include<unistd.h>

#define SERV_TCP_PORT 3434

#define max 60

void stuff();

int i,j,tem;

char buff[4096],t;

FILE *f1;
```

```
int main(int arg,char*argv)
{
int sockfd,newsockfd,clength;
struct sockaddr_in serv_addr,cli_addr;
char t[max],str[max];
strcpy(t,"exit");
sockfd=socket(AF__INET,SOCK_STREAM,0);
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=INADDR_ANY;
serv_addr.sin_port=htons(SERV_TCP_PORT);
bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
listen(sockfd,5);for(;;)
{ clength=sizeof(cli_adr);
new sockfd=accept(sockfd,(struct sockaddr *)&cli_addr,&clength);
close(sockfd);
read(newsockfd,&str,max);
printf("\n CLIENT MESSAFE \n FILENAME: %s \n",str);
f1=fopen(str,"r");
while(fgets(buff,4096,f1)!=NULL)
{ stuff();
write(newsockfd, buff,max);
printf("\n");
}
fclose(f1);
exit(0);
}
printf("\n file transferred \n);
return 0;
}
void stuff()
```

```
{
i=j=tem=0;
printf("\n Inout %s \n",buff);
do
{
if(buff[i]=='1')
{
if(j==4)
{
tem=(i-4);
t='0';
for(;tem<=i;tem++)
printf("%c",buff[tem]);
j=-1;
printf("%c",t);
}
i+=1;
j+=1;
}
else
{
tem=I;
tem=tem-j;
for(:tem<=i,tem++)
printf("%c",buff[tem]);
i+=1;
j=0;
}
}
while(buff[i]!='\0');
}
```

SAMPLE OUTPUT :

```
CLIENT MESSAGE
FILENAME :source.txt
Input 1111111111111
111110111110
```

CLIENT

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#include<fcntl.h>
#include<stdlib.h>
#include<unistd.h>
#define SERV_TCP_PORT 3434
#define max 60
int main(int argc,char *argv[])
{
int sockfd;
struct sockaddr_in serv_addr;
struct hostent * server;
char send[max], recv[max],s[max],name[max];
sockfd=socket(AF_INET,SOCK_STREAM,0);
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
serv_addr.sin_port=htons(SERV_TCP_PORT);
connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));
for(;;)
{
printf("\n source filename :\t");
scanf("%s,send);
write(sockfd,send,max);
while(1)
{
read(sockfd,recv,max);
}
exit(0);
}
close(sockfd);
return ;
}
```

SAMPLE OUTPUT

```
Source filename:  source.txt
```

RESULT:

Thus a c program for bit stuffing is executed and the output is verified successfully.

EX No 12 Congestion Avoidance using Random Early Detection

Aim

To implement Congestion avoidance technique using Random Early Detection (RED)

ALGORITHM

1. RED provides congestion avoidance by controlling the queue size at the gateway.
2. RED notifies the source before the congestion actually happens rather than wait till it actually occurs.
3. RED provides a mechanism for the gateway to provide some feedback to the source on congestion status.
4. For Each incoming packet

If AvgQ <= minQ

queue packet

If minQ <= AvgQ < maxQ

Mark packet with probability P

If maxQ <= AvgQ

Mark the packet

5. For Each incoming packet

If AvgQ <= minQ

queue packet

If minQ <= AvgQ < maxQ

Mark packet with probability P

If maxQ <= AvgQ

Mark the packet

6. TempP = MaxP x (AvgQ – minQ)/(maxQ – minQ)
7. P = TempP/ (1 – count.TempP)
    a. Count is number of newly arrived packets queued.

b. This Extra step has been introduced to keep the drops evenly spaced in time.

```
#include <stdio.h>
#include<conio.h>
int main()
{
int n;
int i,j,k;
int a[10][10],b[10][10];
printf("\n Enter the number of nodes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("\n Enter the distance between the host %d - %d:",i+1,j+1);
scanf("%d",&a[i][j]);
}}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
for(k=0;k<n;k++)
{
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(a[i][j]>a[i][k]+a[k][j])
{
a[i][j]=a[i][k]+a[k][j];
}}}}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
b[i][j]=a[i][j];
if(i==j)
{
b[i][j]=0;
}
}}
printf("\n The output matrix:\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
getch();
}
```

Output:

```
Enter the number of nodes:4
Enter the distance between the host 1 - 1:5
Enter the distance between the host 1 - 2:9
Enter the distance between the host 1 - 3:6
Enter the distance between the host 1 - 4:4
Enter the distance between the host 2 - 1:2
Enter the distance between the host 2 - 2:1
Enter the distance between the host 2 - 3:8
Enter the distance between the host 2 - 4:3
Enter the distance between the host 3 - 1:6
Enter the distance between the host 3 - 2:1
Enter the distance between the host 3 - 3:4
Enter the distance between the host 3 - 4:2
Enter the distance between the host 4 - 1:5
Enter the distance between the host 4 - 2:1
Enter the distance between the host 4 - 3:8
Enter the distance between the host 4 - 4:2
5 9 6 4
2 1 8 3
6 1 4 2
5 1 8 2

The output matrix:
0 5 6 4
2 0 8 3
3 1 0 2
3 1 8 0
```

RESULT:

Thus the above program to simulate the Routing Protocols using border gateway protocol is executed and the output is verified successfully.