MAHA BARATHI ENGINEERING COLLEGE
CHINNASALEM – 606 201

Department of Computer Science and Engineering

SUBJECT CODE / NAME	:	CS 3591 / NETWORKS LABORATORY
YEAR/ SEMESTER	:	III/ V
REGULATION	:	2021

PREPARED BY
Mr. N. Khadirkumar (HOD / CSE)

APPROVED BY
Mr. N. Khadirkumar (HOD / CSE)

SYLLABUS

PRACTICAL EXERCISES:

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

2. Write a HTTP web client program to download a web page using TCP sockets.

3. Applications using TCP sockets like: a) Echo client and echo server b) Chat

4. Simulation of DNS using UDP sockets.

5. Use a tool like Wireshark to capture packets and examine the packets

6. Write a code simulating ARP /RARP protocols.

7. Study of Network simulator (NS) and Simulation of Congestion Control Algorithms usingNS.

8. Study of TCP/UDP performance using Simulation tool.

9. Simulation of Distance Vector/ Link State Routing algorithm.

10. Simulation of an error correction code (like CRC)

Total : 30 PERIODS

TABLE OF CONTENTS

EX.NO. 1A

COMMANDS - TCPDUMP, NETSTAT, IFCONFIG, NSLOOKUP AND TRACEROUTE

Aim:

To use commands like tcpdump, netstat, ifconfig, nslookup and traceroute.

1. Tcpdump

Tcpdump is a command line utility that allows you to capture and analyze network traffic going through your system.

Procedure:

Check if tcpdump is installed on your system
$ which tcpdump
/usr/sbin/tcpdump
If tcpdump is not installed,
$ sudo apt install tcpdump
To get Supervisor Privilege
$ sudo –i to change #
(and password vrsoopslab)
($ is changed to # and the commands can be executed in supervisor)

Capturing packets with tcpdump

- Use the command tcpdump -D to see which interfaces are available for capture.

root@ubuntu:~# tcpdump -D

1. ens33 [Up, Running] // The ens33 network interface has the IPv4 address
2. lo [Up, Running, Loopback]
3. any (Pseudo-device that captures on all interfaces) [Up, Running]
4. bluetooth-monitor (Bluetooth Linux Monitor) [none]
5. nflog (Linux netfilter log (NFLOG) interface) [none]
6. nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]

Capture all packets in any interface by running this command:

root@ubuntu:~# tcpdump -i any

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
21:14:21.943503 ARP, Request who-has _gateway tell 192.168.47.1, length 46
21:14:21.946006 IP localhost.43779 > localhost.domain: 58970+ [1au] PTR? 2.47.168.192.in-addr.arpa. (54)
21:14:21.947049 IP ubuntu.38091 > _gateway.domain: 20883+ [1au] PTR? 2.47.168.192.in-addr.arpa. (54)
21:14:22.969221 ARP, Request who-has _gateway tell **192.168.47.**1, length 46
21:14:23.934408 ARP, Request who-has _gateway tell 192.168.47.1, length 46
21:14:24.931185 ARP, Request who-has _gateway tell 192.168.47.1, length 46
21:14:25.974965 ARP, Request who-has _gateway tell 192.168.47.1, length 46

21:14:26.082444 IP localhost.57652 > localhost.domain: 57276+ [1au] PTR? 53.0.0.127.in-addr.arpa. (52)
^C
8 packets captured
33 packets received by filter
22 packets dropped by kernel

Filter packets based on the source and destination IP Address:

root@ubuntu:~# tcpdump -i any -c5 -nn src 192.168.47.1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
21:20:21.945641 ARP, Request who-has 192.168.47.2 tell 192.168.47.1, length 46
21:20:22.988652 ARP, Request who-has 192.168.47.2 tell 192.168.47.1, length 46
21:20:23.938980 ARP, Request who-has 192.168.47.2 tell 192.168.47.1, length 46
21:20:24.936109 ARP, Request who-has 192.168.47.2 tell 192.168.47.1, length 46
21:20:25.993610 ARP, Request who-has 192.168.47.2 tell 192.168.47.1, length 46
5 packets captured
5 packets received by filter
0 packets dropped by kernel

root@ubuntu:~# tcpdump -i any -c5 -nn dst 192.168.47.2

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
21:21:21.481064 ARP, Request who-has 192.168.47.2 tell 192.168.47.1, length 46
21:21:22.440157 ARP, Request who-has 192.168.47.2 tell 192.168.47.1, length 46
21:21:23.438088 ARP, Request who-has 192.168.47.2 tell 192.168.47.1, length 46
21:21:24.498737 ARP, Request who-has 192.168.47.2 tell 192.168.47.1, length 46
21:21:25.435089 ARP, Request who-has 192.168.47.2 tell 192.168.47.1, length 46
5 packets captured
5 packets received by filter
0 packets dropped by kernel

2. netstat

netstat (network statistics) is a command line tool for monitoring network connections both incoming and outgoing as well as viewing routing tables, interface statistics etc.

-at → list all TCP ports
-au → list all UDP ports
-l → listening ports
-lt → listening TCP
-lu → listening UDP
-s → statistics of all ports
-su →statistics of UDP
-st → statistics of TCP

root@ubuntu:~# netstat

Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
udp 0 0 ubuntu:bootpc 192.168.47.254:bootps ESTABLISHED
Active UNIX domain sockets (w/o servers)

3. ifconfig

It displays the details of a network interface card like IP address, MAC Address, and the status of a network interface card.

root@ubuntu:~# ifconfig

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.47.128 netmask 255.255.255.0 broadcast 192.168.47.255
inet6 fe80::47b1:5882:c339:f3c3 prefixlen 64 scopeid 0x20<link>
ether 00:0c:29:21:f4:b0 txqueuelen 1000 (Ethernet)
RX packets 73806 bytes 109865283 (109.8 MB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 11137 bytes 687838 (687.8 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 202 bytes 17315 (17.3 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 202 bytes 17315 (17.3 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

4. nslookup

nslookup (stands for "Name Server Lookup") is a useful command for getting information from DNS server. It is a network administration tool for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or any other specific DNS record.

root@ubuntu:~# nslookup google.com

root@ubuntu:~# nslookup google.com
Server: 127.0.0.53
Address: 127 0 0.53#53

Non-authoritative answer:
Name: google.com
Address: 142.250.183.14
Name: google.com
Address: 2404:6800:4009:820::200e

root@ubuntu:~# nslookup 142.250.183.14

root@ubuntu:~# nslookup 142.250.183.14
14.183.250.142.in-addr.arpa name = bom07s30-in-f14.1e100.net.
Authoritative answers can be found from:

Lookup for any record

root@ubuntu:~# nslookup -type=any google.com

root@ubuntu:~# nslookup -type=any google.com
Server: 127.0.0.53
Address: 127 0 0.53#53

Non-authoritative answer:
Name: google.com
Address: 142.250.183.14
Name: google.com
Address: 2404:6800:4009:820::200e
google.com text = "facebook-domain-verification=22rm551cu4k0ab0bxsw536tlds4h95"
google.com nameserver = ns1.google.com.
google.com nameserver = ns4.google.com.
google.com text = "apple-domain-verification=30afIBcvSuDV2PLX"
google.com text = "docusign=1b0a6754-49b1-4db5-8540-d2c12664b289"
google.com text = "google-site-verification=wD8N7i1JTNTkezJ49swvWW48f8_9xveREV4oB-0Hf5o"
google.com rdata_257 = 0 issue "pki.goog"
google.com
origin = ns1.google.com
mail addr = dns-admin.google.com
serial = 551157852
refresh = 900
retry = 900
expire = 1800
minimum = 60

Lookup for an ns record

root@ubuntu:~# nslookup -type=ns google.com
Server: 127.0.0.53
Address: 127 0 0.53#53

Non-authoritative answer:
google.com nameserver = ns3.google.com.
google.com nameserver = ns1.google.com.
google.com nameserver = ns4.google.com.
google.com nameserver = ns2.google.com.
Authoritative answers can be found from:

5. traceroute

The traceroute command is used in Linux to map the journey that a packet of information undertakes from its source to its destination.

root@ubuntu:~# traceroute

Usage:

traceroute [-46dFITnreAUDV] [-f first_ttl] [-g gate,...] [-i device] [-m max_ttl] [-N squeries] [-p port] [-t tos] [-l flow_label] [-w MAX,HERE,NEAR] [-q nqueries] [-s src_addr] [-z sendwait] [--fwmark=num] host [packetlen]

Options:

-4 Use IPv4

-6 Use IPv6

-d --debug Enable socket level debugging

-F --dont-fragment Do not fragment packets

root@ubuntu:~# traceroute google.com

traceroute to google.com (142.250.183.14), 30 hops max, 60 byte packets

1 _gateway (192.168.47.2) 1.636 ms 1.526 ms 1.456 ms

2 * * *

3 * * *

4 * * *

5 * * *

6 * * *

7 * * *

8 * * *

9 * * *

10 * * *

11 * * *

12 * *^C

root@ubuntu:~# traceroute 142.250.183.12

traceroute to 142.250.183.12 (142.250.183.12), 30 hops max, 60 byte packets

1 _gateway (192.168.47.2) 1.181 ms 1.122 ms 1.077 ms

2 * * *

3 * * *

4 * * *

5 * * *

6 * * *

7 * * *

8 * * *

9 * *^C

Result:

Thus the uses of commands are studied and implemented successfully.

EX.NO: 1B

CAPTURE PING AND TRACEROUTE PDUS USING A NETWORK PROTOCOL ANALYZER

Aim:

To use the commands Ping and traceroute Capture ping and traceroute PDUs using a network protocol analyzer and examine.

1. ping Command

- PING (Packet Internet Groper) command is used to check the network connectivity between host and server/host.
- This command takes as input the IP address or the URL and sends a data packet to the specified address with the message "PING" and get a response from the server/host this time is recorded which is called latency (Delay).

Procedure:

root@ubuntu:~# ping www.google.com

PING www.google.com (142.250.205.228) 56(84) bytes of data.
64 bytes from maa05s28-in-f4.1e100.net (142.250.205.228): icmp_seq=1 ttl=128 time=17.9 ms
64 bytes from maa05s28-in-f4.1e100.net (142.250.205.228): icmp_seq=2 ttl=128 time=11.0 ms
64 bytes from maa05s28-in-f4.1e100.net (142.250.205.228): icmp_seq=3 ttl=128 time=48.3 ms
64 bytes from maa05s28-in-f4.1e100.net (142.250.205.228): icmp_seq=4 ttl=128 time=10.6 ms
64 bytes from maa05s28-in-f4.1e100.net (142.250.205.228): icmp_seq=5 ttl=128 time=10.8 ms
64 bytes from maa05s28-in-f4.1e100.net (142.250.205.228): icmp_seq=8 ttl=128 time=11.0 ms
64 bytes from maa05s28-in-f4.1e100.net (142.250.205.228): icmp_seq=9 ttl=128 time=10.7 ms
--- www.google.com ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8015ms
rtt min/avg/max/mdev = 10.555/15.855/48.307/11.682 ms

root@ubuntu:~# ping 192.168.30.68

PING 192.168.30.68 (192.168.30.68) 56(84) bytes of data.
64 bytes from 192.168.30.68: icmp_seq=1 ttl=128 time=1.65 ms
64 bytes from 192.168.30.68: icmp_seq=2 ttl=128 time=1.66 ms
64 bytes from 192.168.30.68: icmp_seq=3 ttl=128 time=1.56 ms
64 bytes from 192.168.30.68: icmp_seq=4 ttl=128 time=1.34 ms
64 bytes from 192.168.30.68: icmp_seq=5 ttl=128 time=1.20 ms
64 bytes from 192.168.30.68: icmp_seq=8 ttl=128 time=1.56 ms
--- 192.168.30.68 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7014ms
rtt min/avg/max/mdev = 1.201/1.515/1.663/0.153 ms

Result:

Thus the ping and traceroute PDUs are captured using a network protocol analyzer has been examined successfully.

EX.NO.2

CREATE A SOCKET FOR HTTP FOR WEB PAGE DOWNLOAD

Aim:

To write a program in java to create a socket for HTTP for web page download.

Algorithm:

1. Start the program
2. Read the file to be downloaded from webpage
3. To download an image, use java URL class which can be found under java.net package.
4. The file is downloaded from server and is stored in the current working directory.
5. Stop the program

Program:

```
import java.io.*;
import java.net.URL;
public class Download
{
	public static void main(String[] args) throws Exception
	{
	try
	{
		String fileName = "digital_image_processing.jpg";
		String website = "http://tutorialspoint.com/java_dip/images/"+fileName;
		System.out.println("Downloading File From: " + website);
		URL url = new URL(website);
		InputStream inputStream = url.openStream();
		OutputStream outputStream = new FileOutputStream(fileName);
		byte[] buffer = new byte[2048];
		int length = 0;
		while ((length = inputStream.read(buffer)) != -1)
		{
			System.out.println("Buffer Read of length: " + length);
			outputStream.write(buffer, 0, length);
		}
		inputStream.close();
		outputStream.close();
	}
		catch(Exception e)
		{
			System.out.println("Exception: " + e.getMessage());
		}
	}
}
```

Output

C:\Program Files\Java\jdk-19\bin>java Download
Downloading File From:
http://tutorialspoint.com/java_dip/images/digital_image_processing.jpg
Buffer Read of length: 256

Result:

Thus created a socket for HTTP in JAVA for web page download.

EX.NO:3A

IMPLEMENTATION OF ECHO CLIENT AND ECHO SERVER USING TCP

Aim:

To implement echo server and client in java using TCP sockets.

Algorithm:

Server:

1. Create a server socket.
2. Wait for client to be connected.
3. Read text from the client
4. Echo the text back to the client.
5. Repeat steps 4-5 until 'bye' or 'null' is read.
6. Close the I/O streams
7. Close the server socket
8. Stop

Client:

1. Create a socket and establish connection with the server
2. Get input from user.
3. If equal to bye or null, then go to step 7.
4. Send text to the server.
5. Display the text echoed by the server
6. Repeat steps 2-4
7. Close the I/O streams
8. Close the client socket
9. Stop

Program1:

```
import java.net.*;
import java.io.*;
public class tcpechoclient
{
	public static void main(String[] args) throws IOException
	{
		BufferedReader fromServer = null, fromUser = null;
		PrintWriter toServer = null;
		Socket sock = null;
		try
		{
		if (args.length == 0)
		sock = new Socket(InetAddress.getLocalHost(),4000);
		else
		sock = new Socket(InetAddress.getByName(args[0]),4000);
		fromServer = new BufferedReader(new InputStreamReader(sock.getInputStream()));
```

```
                fromUser = new BufferedReader(new InputStreamReader(System.in));
                toServer = new PrintWriter(sock.getOutputStream(),true);
                String Usrmsg, Srvmsg;
                System.out.println("Type \"bye\" to quit");
                while (true)
                {
                        System.out.print("Enter msg to server : ");
                        Usrmsg = fromUser.readLine();
                        if (Usrmsg==null || Usrmsg.equals("bye"))
                        {
                                toServer.println("bye");
                                break;
                        }
                        else
                                toServer.println(Usrmsg);
                                Srvmsg = fromServer.readLine();
                                System.out.println(Srvmsg);
                }
                fromUser.close();
                fromServer.close();
                toServer.close();
                sock.close();
                }
        catch (IOException ioe)
        {
                System.err.println(ioe);
        }
        }
}
```

Program2:

```
import java.net.*;
import java.io.*;
public class tcpechoserver
{
        public static void main(String[] arg) throws IOException
        {
                ServerSocket sock = null;
                BufferedReader fromClient = null;
                OutputStreamWriter toClient = null;
                Socket client = null;
                try
                {
                sock = new ServerSocket(4000);
                System.out.println("Server Ready");
```

```
                client = sock.accept();
                System.out.println("Client Connected");
                fromClient = new BufferedReader(new
                InputStreamReader(client.getInputStream()));
                toClient = new OutputStreamWriter(client.getOutputStream());
                String line;
                while (true)
                {
                line = fromClient.readLine();
                if ( (line == null) || line.equals("bye"))
                break;
                System.out.println ("Client [ " + line + " ]");
                toClient.write("Server [ "+ line +" ]\n");
                toClient.flush();
                }
                fromClient.close();
                toClient.close();
                client.close();
                sock.close();
                System.out.println("Client Disconnected");
                }
                catch (IOException ioe)
                {
                        System.err.println(ioe);
                }
        }
}
```

Output:

Server

```
C:\Program Files\Java\jdk-19\bin>javac tcpechoserver.java
C:\Program Files\Java\jdk-19\bin>java tcpechoserver
Server Ready
Client Connected
Client [  ]
Client [ hi ]
Client [ hello ]
Client Disconnected
```

Client

```
C:\Program Files\Java\jdk-19\bin>javac tcpechoclient.java
C:\Program Files\Java\jdk-19\bin>java tcpechoclient
Type "bye" to quit
Enter msg to server :
Server [ ]
```

Enter msg to server : hi
Server [hi]
Enter msg to server : hello
Server [hello]
Enter msg to server : bye

RESULT:

Thus, data from client to server is echoed back to the client to check reliability/noise level of the channel.

EX.NO:3B

IMPLEMENTATION OF CHAT USING TCP

Aim:

To implement a chat server and client in java using TCP sockets.

Algorithm:

Server:

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

Client:

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

Program1:

```
import java.io.*;
import java.net.*;
class tcpchatserver
{
public static void main(String args[])throws Exception
{
        PrintWriter toClient;
        BufferedReader fromUser, fromClient;
        try
        {
                ServerSocket Srv = new ServerSocket(5555);
                System.out.print("\nServer started\n");
                Socket Clt = Srv.accept();
                System.out.println("Client connected");
                toClient = new PrintWriter(new BufferedWriter(new
                OutputStreamWriter(Clt.getOutputStream())), true);
                fromClient = new BufferedReader(new
                InputStreamReader(Clt.getInputStream()));
```

```
                fromUser = new BufferedReader(new
                InputStreamReader(System.in));
                String CltMsg, SrvMsg;
                while(true)
                {
                        CltMsg= fromClient.readLine();
                        if(CltMsg.equals("end"))
                        break;
                        else
                        {
                                System.out.println("\nServer <<< " +CltMsg);
                                System.out.print("Message to Client : ");
                                SrvMsg = fromUser.readLine();
                                toClient.println(SrvMsg);
                        }
                }
                System.out.println("\nClient Disconnected");
                fromClient.close();
                toClient.close();
                fromUser.close();
                Clt.close();
                Srv.close();
        }
        catch (Exception E)
        {
                System.out.println(E.getMessage());
        }
}
}
```

Program2:

```
import java.io.*;
import java.net.*;
class tcpchatclient
{
public static void main(String args[])throws Exception
{
        Socket Clt;
        PrintWriter toServer;
        BufferedReader fromUser, fromServer;
        try
        {
                if (args.length > 1)
                {
                        System.out.println("Usage: java hostipaddr");
```

```
                System.exit(-1);
            }
            if (args.length == 0)
            Clt = new Socket(InetAddress.getLocalHost(),5555);
            else
            Clt = new Socket(InetAddress.getByName(args[0]),5555);
            toServer = new PrintWriter(new BufferedWriter(new
            OutputStreamWriter(Clt.getOutputStream())), true);
            fromServer = new BufferedReader(new
            InputStreamReader(Clt.getInputStream()));
            fromUser = new BufferedReader(new
            InputStreamReader(System.in));
            String CltMsg, SrvMsg;
            System.out.println("Type \"end\" to Quit");
            while (true)
            {
                System.out.print("\nMessage to Server : ");
                CltMsg = fromUser.readLine();
                toServer.println(CltMsg);
                if (CltMsg.equals("end"))
                break;
                SrvMsg = fromServer.readLine();
                System.out.println("Client <<< " + SrvMsg);
            }
        }
        catch(Exception E)
        {
            System.out.println(E.getMessage());
        }
}
}
```

Output:

Server

```
C:\Program Files\Java\jdk-19\bin>javac tcpchatserver.java
C:\Program Files\Java\jdk-19\bin>java tcpchatserver

Server started
Client connected

Server <<< hi
Message to Client : hello

Server <<< how are you
Message to Client : im fine
```

Server <<< quit
Message to Client : quit

Client Disconnected

Client

C:\Program Files\Java\jdk-19\bin>javac tcpchatclient.java
C:\Program Files\Java\jdk-19\bin>java tcpchatclient
Type "end" to Quit

Message to Server : hi
Client <<< hello

Message to Server : how are you
Client <<< im fine

Message to Server : quit
Client <<< quit

Message to Server : end

RESULT:

Thus both the client and server exchange data using TCP socket programming.

EX.NO:4

IMPLEMENTATION OF DNS SOCKET

Aim:

To implement a DNS server and client in java using UDP sockets.

Algorithm:

Server:

1. Define a array of hosts and its corresponding IP address in another array
2. Create a datagram socket
3. Create a datagram packet to receive client request
4. Read the domain name from client to be resolved
5. Lookup the host array for the domain name
6. If found then retrieve corresponding address
7. Construct a datagram packet to send response back to the client
8. Repeat steps 3-7 to resolve further requests from clients
9. Close the server socket
10. Stop

Client:

1. Create a datagram socket
2. Get domain name from user
3. Construct a datagram packet to send domain name to the server
4. Create a datagram packet to receive server message
5. If it contains IP address then display it, else display "Domain does not exist"
6. Close the client socket
7. Stop

Program1 :

```
import java.io.*;
import java.net.*;
public class udpdnsclient
{
public static void main(String args[])throws IOException
{
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientsocket = new DatagramSocket();
        InetAddress ipaddress;
        if (args.length == 0)
        ipaddress = InetAddress.getLocalHost();
        else
        ipaddress = InetAddress.getByName(args[0]);
        byte[] senddata = new byte[1024];
        byte[] receivedata = new byte[1024];
        int portaddr = 1362;
```

```
        System.out.print("Enter the hostname : ");
        String sentence = br.readLine();
        senddata = sentence.getBytes();
        DatagramPacket pack = new DatagramPacket(senddata,
        senddata.length, ipaddress,portaddr);
        clientsocket.send(pack);
        DatagramPacket recvpack =new DatagramPacket(receivedata,receivedata.length);
        clientsocket.receive(recvpack);
        String modified = new String(recvpack.getData());
        System.out.println("IP Address: " + modified);
        clientsocket.close();
        }
}
```

Program2:

```
import java.io.*;
import java.net.*;
public class udpdnsserver
{
        private static int indexOf(String[] array, String str)
        {
        str = str.trim();
        for (int i=0; i < array.length; i++)
        {
        if (array[i].equals(str))
        return i;
        }
        return -1;
        }
        public static void main(String arg[])throws IOException
        {
        String[] hosts = {"yahoo.com", "gmail.com","cricinfo.com", "facebook.com"};
        String[] ip = {"68.180.206.184", "209.85.148.19","80.168.92.140", "69.63.189.16"};
        System.out.println("Press Ctrl + C to Quit");
        while (true)
        {
                DatagramSocket serversocket=new DatagramSocket(1362);
                byte[] senddata = new byte[1021];
                byte[] receivedata = new byte[1021];
                DatagramPacket recvpack = new DatagramPacket(receivedata, receivedata.length);
                serversocket.receive(recvpack);
                String sen = new String(recvpack.getData());
                InetAddress ipaddress = recvpack.getAddress();
                int port = recvpack.getPort();
                String capsent;
```

```
                System.out.println("Request for host " + sen);
                if(indexOf (hosts, sen) != -1)
                capsent = ip[indexOf (hosts, sen)];
                else
                capsent = "Host Not Found";
                senddata = capsent.getBytes();
                DatagramPacket pack = new DatagramPacket(senddata,
                senddata.length,ipaddress,port);
                serversocket.send(pack);
                serversocket.close();
        }
}
}
```

Output:

Server

```
C:\Program Files\Java\jdk-19\bin>javac udpdnsserver.java
C:\Program Files\Java\jdk-19\bin>java udpdnsserver
Press Ctrl + C to Quit
Request for host gmail.com
Request for host youtube.com
```

Client

```
C:\Program Files\Java\jdk-19\bin>javac udpdnsclient.java
C:\Program Files\Java\jdk-19\bin>java udpdnsclient
Enter the hostname : gmail.com
IP Address: 209.85.148.19

C:\Program Files\Java\jdk-19\bin>java udpdnsclient
Enter the hostname : youtube.com
IP Address: Host Not Found
```

RESULT:

Thus domain name requests by the client are resolved into their respective logical address using lookup method.

EX.NO:5

WIRESHARK TO CAPTURE PACKETS AND EXAMINE THE PACKETS

Aim:

To capture and examine the packets using wireshark.

Procedure:

Network protocol analyzer - wireshark

sudo apt install wireshark

- Wireshark is free & Open source network packet analyzer that is used for network analysis,
- troubleshooting, etc.
- Wireshark is quite similar to tcpdump, the major difference between the two is that Wireshark has a graphical interface with built-in filtering options, which make it easy to use.

Installation commands on Wireshark

sudo apt install wireshark

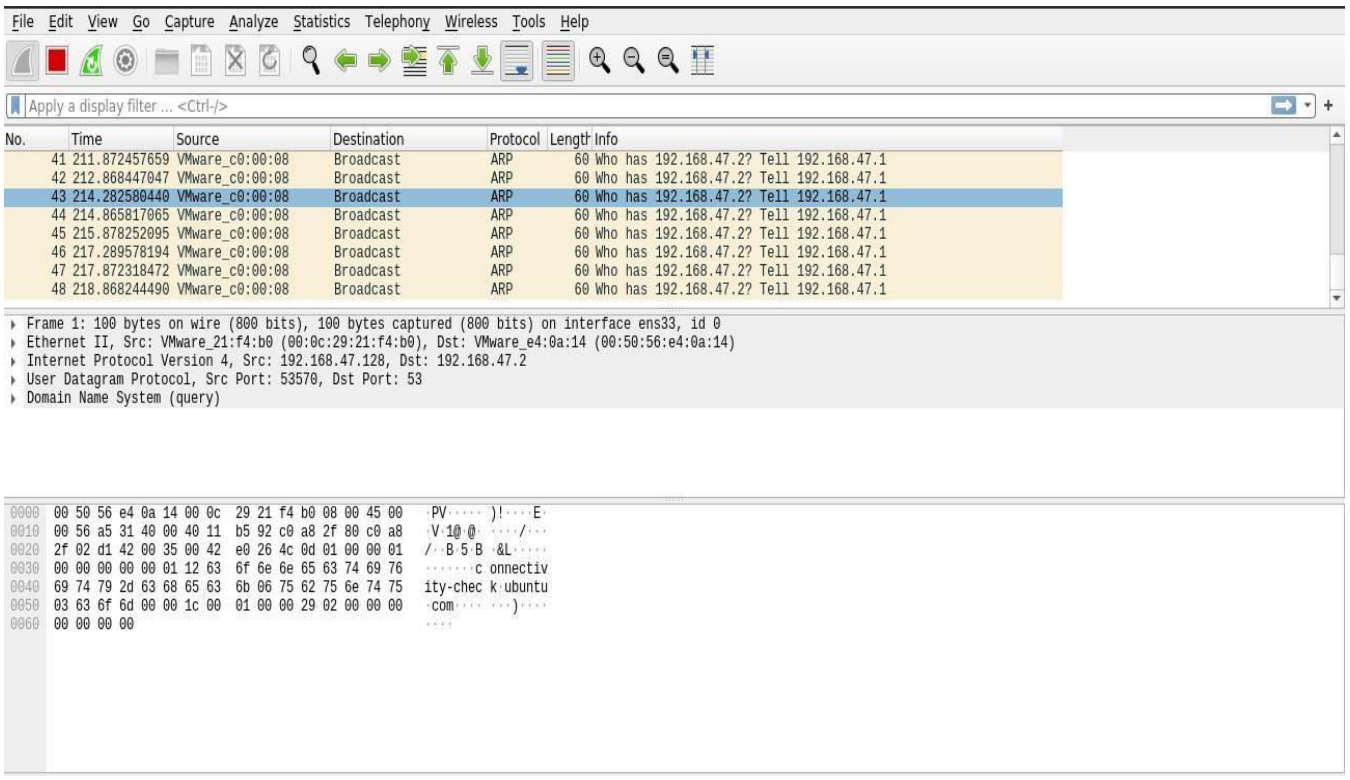
To Open Wireshark

Open directly or use the following commands

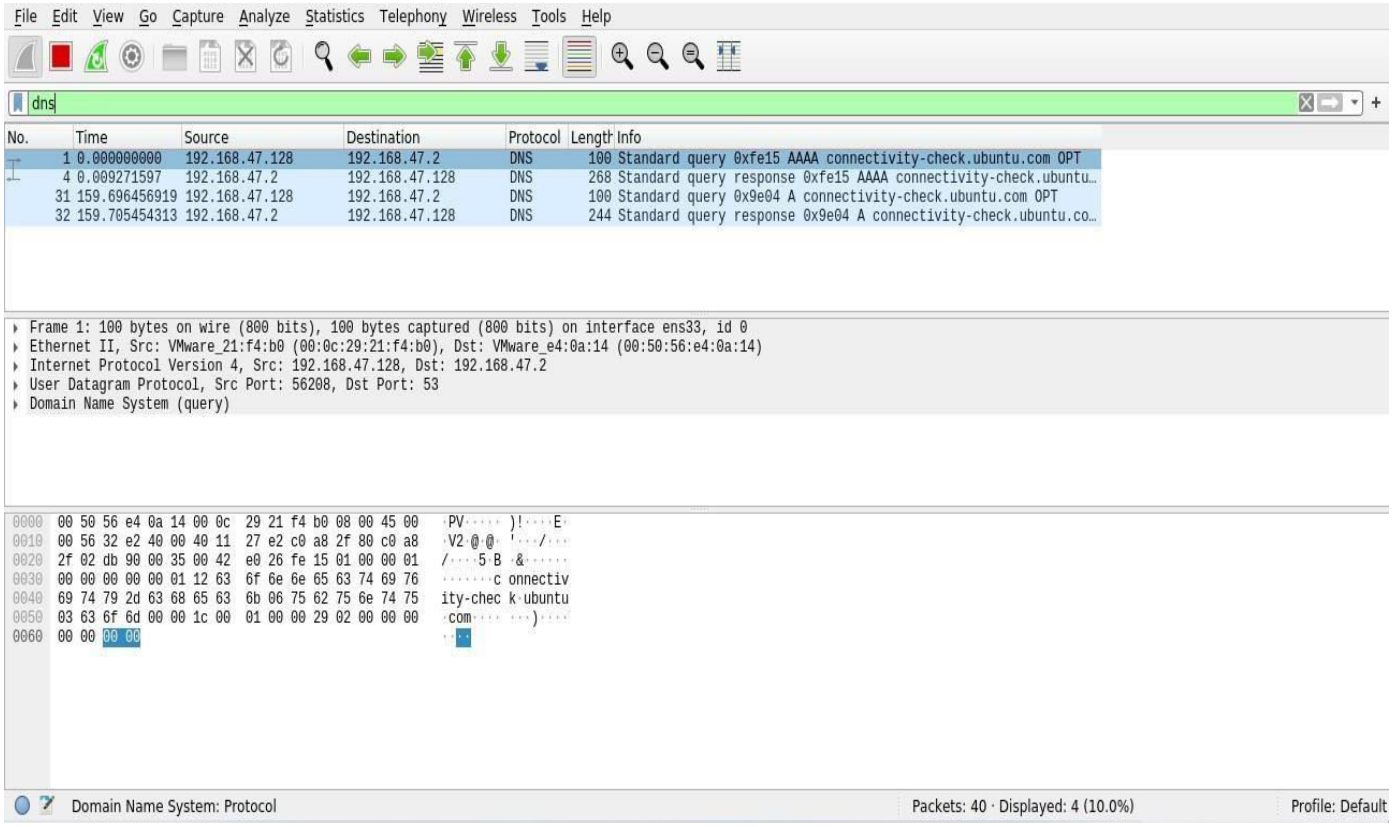
sudo wireshark

root@ubuntu:~# sudo wireshark

QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'



Filter:



Result:

Thus a packets are capture and examine successfully.

EX.NO: 6A

IMPLEMENTATION OF ARP PROTOCOL

Aim:

To Implement the ARP protocol using java.

Algorithm:

server:

1. Create a sender socket.
2. Wait for client to be connected.
3. Enter the IP address
4. Send IP address to the client
5. Close the I/O streams
6. Close the server socket
7. Stop

Client:

1. Create a socket and establish connection with the server
2. Get IP address from server.
3. Convert into physical address.
4. Close the I/O streams
5. Close the client socket
6. Stop

Program 1:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverarp
{
	public static void main(String args[])
	{
	try
	{
		ServerSocket obj=new ServerSocket(130);
		Socket obj1=obj.accept();
		while(true)
		{
			DataInputStream din=new DataInputStream(obj1.getInputStream());
			DataOutputStream dout=new
			DataOutputStream(obj1.getOutputStream());
			String str=din.readLine();
			String ip[]={"165.165.80.80","165.165.79.1"};
			String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
```

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

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

Program 2:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientarp
{
    public static void main(String args[])
    {
    try
    {
        BufferedReader in=new BufferedReader(new
        InputStreamReader(System.in));
        Socket clsct=new Socket("127.0.0.1",130);
        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\Java\jdk-19\bin>java Serverarp
```

Client

```
C:\Program Files\Java\jdk-19\bin>javac Clientarp.java
C:\Program Files\Java\jdk-19\bin>java Clientarp
Enter the Logical address(IP):
165.165.80.80
The Physical Address is: 6A:08:AA:C2
```

RESULT:

Thus the ARP Protocol is implemented and IP address is converted into physical address.

EX.NO: 6B

IMPLEMENTATION OF RARP PROTOCOL

Aim:

To Implement the Protocol that convert physical address into IP address using java.

Algorithm:

Server:

1. Create a sender socket.
2. Wait for client to be connected.
3. Enter the Physical address
4. Send Physical address to the client
5. Close the I/O streams
6. Close the server socket
7. Stop

Client:

1. Create a socket and establish connection with the server
2. Get Physical address from server.
3. Convert into IP address.
4. Close the I/O streams
5. Close the client socket
6. Stop

Program 1:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverrarp
{
	public static void main(String args[])
	{
	try
	{
		ServerSocket obj=new ServerSocket(130);
		Socket obj1=obj.accept();
		while(true)
		{
			DataInputStream din=new DataInputStream(obj1.getInputStream());
			DataOutputStream dout=new
			DataOutputStream(obj1.getOutputStream());
			String str=din.readLine();
			String ip[]={"165.165.80.80","165.165.79.1"};
			String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
			for(int i=0;i<mac.length;i++)
```

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

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

Program 2:

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientrarp
{
        public static void main(String args[])
        {
        try
        {
                BufferedReader in=new BufferedReader(new
                InputStreamReader(System.in));
                Socket clsct=new Socket("127.0.0.1",130);
                DataInputStream din=new DataInputStream(clsct.getInputStream());
                DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
                System.out.println("Enter the Physical Addres (MAC):");
                String str1=in.readLine();
                dout.writeBytes(str1+'\n');
                String str=din.readLine();
                System.out.println("The Logical address is(IP): "+str);
                clsct.close();
        }
        catch (Exception e)
        {
        System.out.println(e);
        }
        }
```

}

OUTPUT

Server

C:\Program Files\Java\jdk-19\bin>java Serverrarp

Client

C:\Program Files\Java\jdk-19\bin>javac Clientrarp.java
Note: Clientrarp.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Program Files\Java\jdk-19\bin>java Clientrarp
Enter the Physical Addres (MAC):
6A:08:AA:C2
The Logical address is(IP): 165.165.80.80

RESULT:

Thus the RARP Protocol is implemented and Physical address is converted into IP address.

Install NS2, NAM & TCL

Open terminal/command window and type the following commands one-by-one:

sudo apt-get update
sudo apt-get install ns2
sudo apt-get install nam
sudo apt-get install tcl

EX.NO: 7A

STUDY OF NETWORK SIMULATOR (NS)

Aim:

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

Congestion Control Algorithms

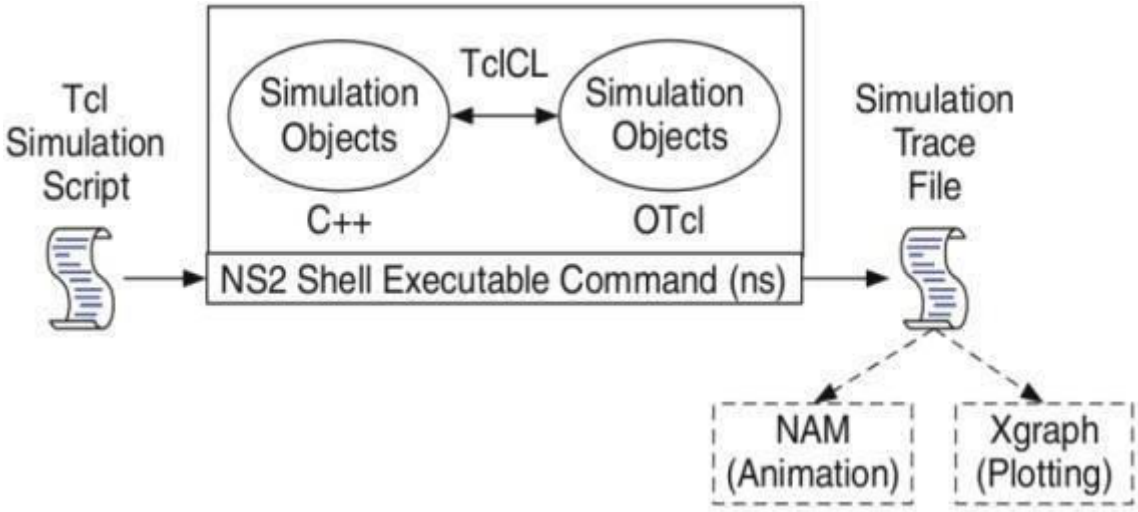
- Slow start
- Additive increase/multiplicative decrease
- Fast retransmit and Fast recovery

Examples of network simulators

There are many both free/open-source and proprietary network simulators. Examples of notable network simulation software are, ordered after how often they are mentioned in research papers:

1. ns (open source)
2. OPNET (proprietary software)
3. NetSim (proprietary software)

Basic Architecture of NS2



Uses of network simulators

- Network simulators serve a variety of needs. Compared to the cost and time involved in setting up an entire test bed containing multiple networked computers, routers and data links, network simulators are relatively fast and inexpensive. They allow engineers, researchers to test scenarios that might be particularly difficult or expensive to emulate using real hardware

- for instance, simulating a scenario with several nodes or experimenting with a new protocol in the network. Network simulators are particularly useful in allowing researchers to test new networking protocols or changes to existing protocols in a controlled and reproducible environment. A typical network simulator encompasses a wide range of networking technologies and can help the users to build complex networks from basic building blocks such as a variety of nodes and links. With the help of simulators, one can design hierarchical networks using various types of nodes like computers, hubs, bridges, routers, switches, links, mobile units etc.

- Various types of Wide Area Network (WAN) technologies like TCP, ATM, IP etc. and Local Area Network (LAN) technologies like Ethernet, token rings etc., can all be simulated with a typical simulator and the user can test, analyze various standard results apart from devising some novel protocol or strategy for routing etc. Network simulators are also widely used to simulate battlefield networks in Network-centric warfare.

- There are a wide variety of network simulators, ranging from the very simple to the very complex. Minimally, a network simulator must enable a user to represent a network topology, specifying the nodes on the network, the links between those nodes and the traffic between the nodes. More complicated systems may allow the user to specify everything about the protocols used to handle traffic in a network. Graphical applications allow users to easily visualize the workings of their simulated environment. Text-based applications may provide a less intuitive interface, but may permit more advanced forms of customization.

BASIC COMMANDS IN NS2

Create event scheduler

```
set ns [new Simulator]
```

Trace packets on all links

```
set nf [open out.nam w]
$ns trace-all $nf
$ns namtrace-all $nf
```

Nodes

```
set n0 [$ns node]
set n1 [$ns node]
```

Links and queuing

```
$ns duplex-link $n0 $n1 <bandwidth> <delay> <queue_type>
<queue_type>: DropTail, RED, etc.
$ns duplex-link $n0 $n1 1Mb 10ms RED
```

Creating a larger topology

```
for {set i 0} {$i < 7} {incr i} {
set n($i) [$ns node]
}
for {set i 0} {$i < 7} {incr i} {
$ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms RED
}
```

Link failures

$ns rtmodel-at <time> up|down $n0 $n1

Creating UDP connection

```
set udp [new Agent/UDP]
set null [new Agent/Null]
$ns attach-agent $n0 $udp
$ns attach-agent $n1 $null
$ns connect $udp $null
```

Creating Traffic (On Top of UDP)

```
set cbr [new Application/Traffic/CBR]
$cbr set packetSize_ 500
$cbr set interval_ 0.005
$cbr attach-agent $udp
```

Post-Processing Procedures

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

Schedule Events

$ns at <time> <event>

Call 'finish'

$ns at 5.0 "finish"

Run the simulation

$ns run

Result:

Thus the study of NS was done successfully

EX.NO.7B

SIMULATION OF CONGESTION CONTROL USING NS2

Aim:

- To simulate the Congestion control algorithm using NS2.

Algorithm:

1. Create a simulation object
2. Set routing protocol to routing
3. Trace packets and all links onto NAM trace and to trace file
4. Create right nodes
5. Describe their layout topology as octagon
6. Add a sink agent to node
7. Connect source and sink

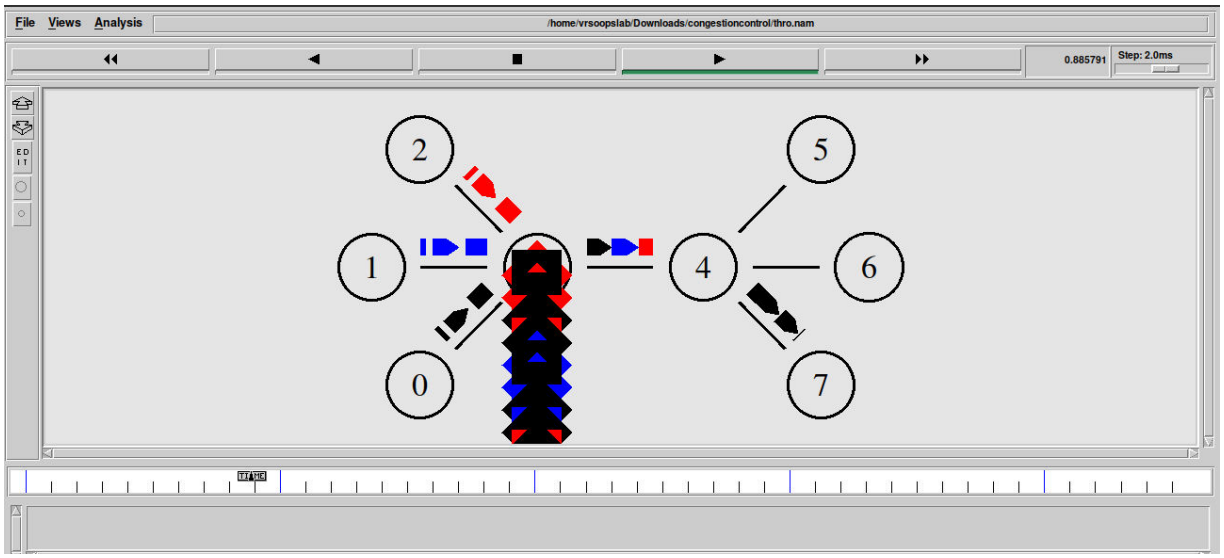
Program:

```
set ns [new Simulator]
set nr [open thro_red.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 $nr
exec nam thro.nam &
exit 0    }
set n0 [$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]
$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
set udp0 [new Agent/UDP]
$ns attach-agent $n2 $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 $n5 $null0
$ns connect $udp0 $null0
set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set  interval_ 0.005
$cbr1 attach-agent $udp1
set 0 [new Agent/Null]
$ns attach-agent $n6 $null0
$ns connect $udp1 $null0
set udp2 [new Agent/UDP]
$ns attach-agent $n0 $udp2
set cbr2 [new Application/Traffic/CBR]
$cbr2 set packetSize_ 500
$cbr2 set  interval_ 0.005
$cbr2 attach-agent $udp2
set null0 [new Agent/Null]
$ns attach-agent $n7 $null0
$ns connect $udp2 $null0
$udp0 set fid_ 1
$udp1 set fid_ 2
$udp2 set fid_ 3
$ns color 1 Red
$ns color 2 Green
$ns color 2 blue
$ns at 0.1 "$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"
```

$ns at 5.0 "finish"
$ns run

Output:



Result:

Thus the study of congestion control algorithm's done successfully.

EX.NO.8A

STUDY OF TCP/UDP PERFORMANCE USING SIMULATION TOOL

Aim:

- To simulate the performance of TCP/UDP using NS2.

Pre lab discussion:

- 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.
- 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.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.

TCP Performance

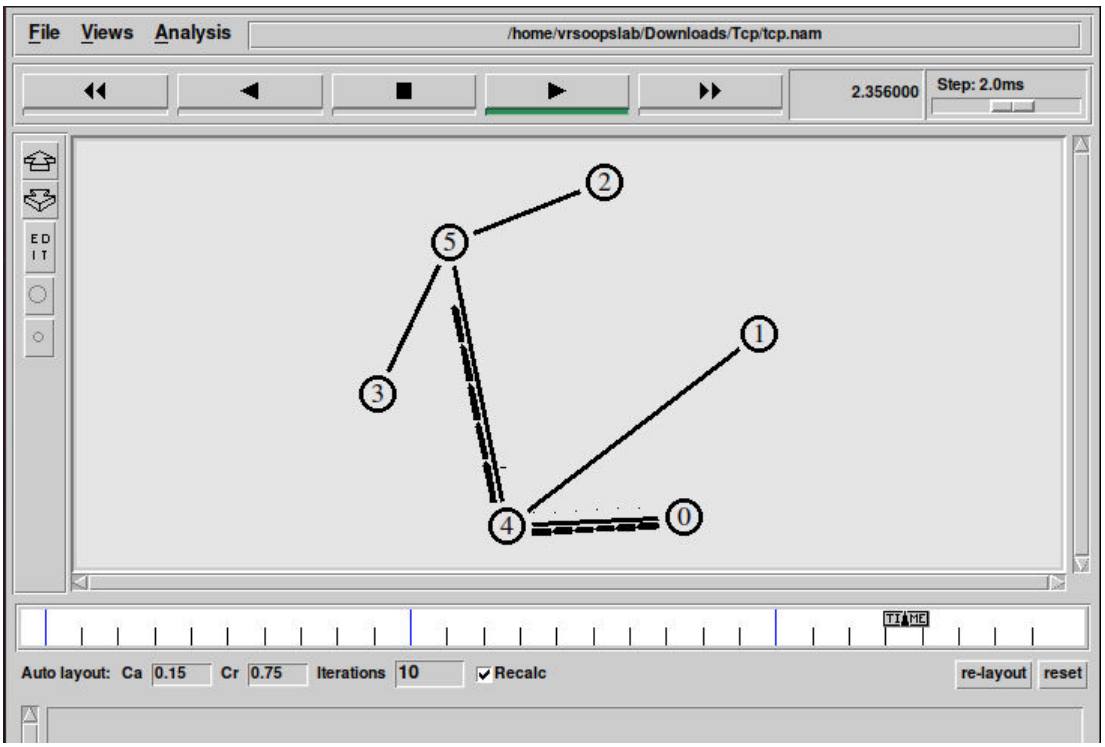
Algorithm:

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

Program:

```
set ns [new Simulator]
set nf [open tcp.nam w]
$ns namtrace-all $nf
set tf [open out.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 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$ns duplex-link $n0 $n4 1Mb 50ms DropTail
$ns duplex-link $n1 $n4 1Mb 50ms DropTail
$ns duplex-link $n2 $n5 1Mb 1ms DropTail
$ns duplex-link $n3 $n5 1Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 50ms DropTail
$ns duplex-link-op $n4 $n5 queuePos 0.5
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n2 $sink
$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.0 "$ftp start"
$ns at 2.5 "$ftp stop"
$ns at 3 "finish"
$ns run
```

Output:



UDP Performance

ALGORITHM :

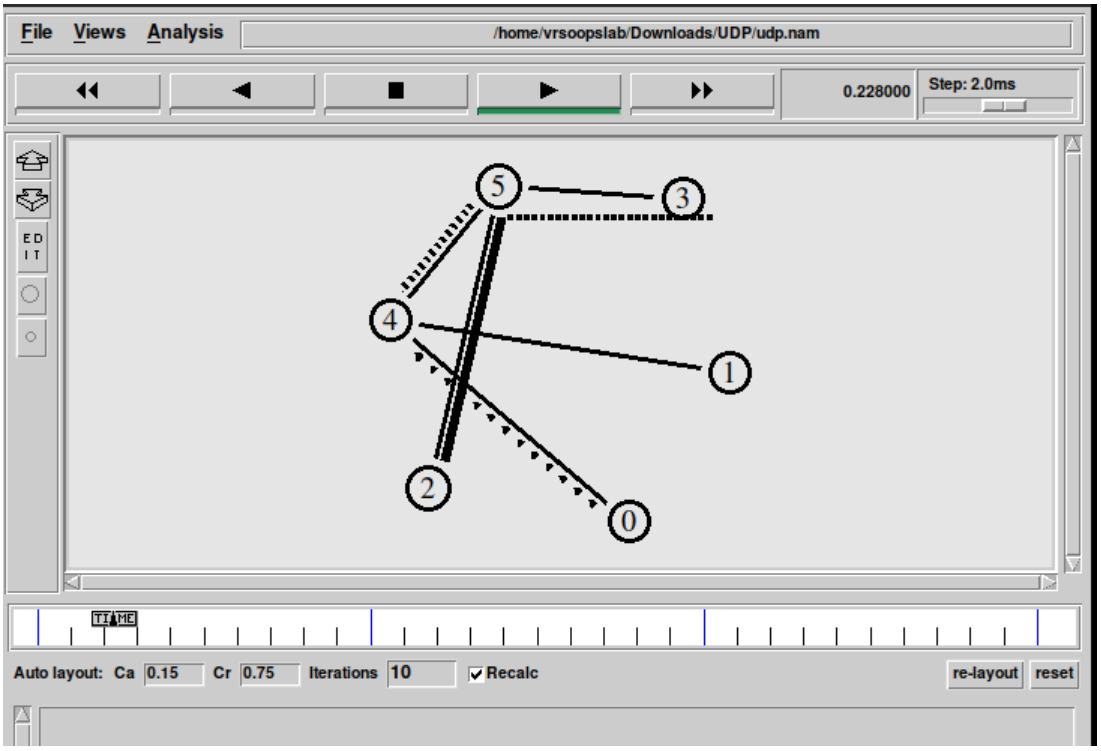
1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.
7. Create the applications and attach them to the UDP agent.
8. Connect udp and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

Program:

```
set ns [new Simulator]
set nf [open udp.nam w]
$ns namtrace-all $nf
set tf [open out.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]
set n4 [$ns node]
set n5 [$ns node]
$ns duplex-link $n0 $n4 1Mb 50ms DropTail
$ns duplex-link $n1 $n4 1Mb 50ms DropTail
$ns duplex-link $n2 $n5 0.1Mb 1ms DropTail
$ns duplex-link $n3 $n5 1Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 50ms DropTail
$ns duplex-link-op $n2 $n5 queuePos 1
set tcp [new Agent/UDP]
$ns attach-agent $n0 $tcp
set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $tcp $sink
set ftp [new Application/Traffic/CBR]
$ftp attach-agent $tcp
$ns at 0.0 "$ftp start"
$ns at 2.5 "$ftp stop"
$ns at 3 "finish"
$ns run
```

Output:



Result :

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

EX.NO.9A

SIMULATION OF DISTANCE VECTOR ROUTING ALGORITHM

Aim:

To simulate the Distance vector and link state routing protocols using NS2.

Algorithm:

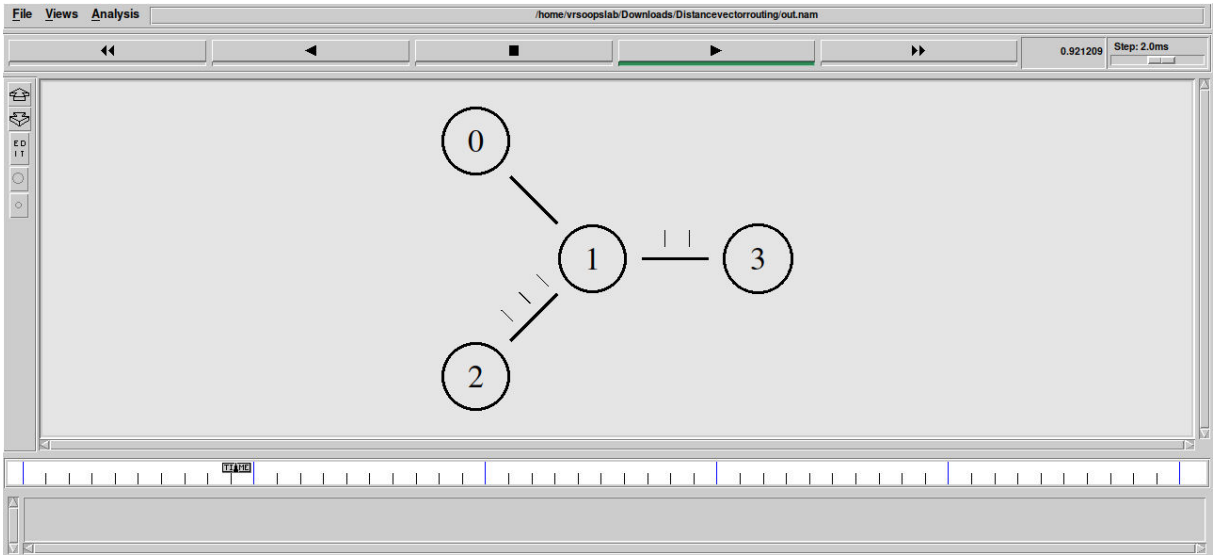
1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.
7. Create the applications and attach them to the UDP/TCP agent.
8. Connect udp/tcp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

Program:

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tr [open out.tr w]
$ns trace-all $tr
proc finish {} {
global nf ns tr
 $ns flush-trace
 close $tr
 exec nam out.nam &
 exit 0
 }
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail
$ns duplex-link-op $n0 $n1 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n1 orient right-up
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set sink [new Agent/TCPSink]
```

```
$ns attach-agent $n3 $sink
set udp [new Agent/UDP]
$ns attach-agent $n2 $udp
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $tcp $sink
$ns connect $udp $null
$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3
$ns rtproto DV
$ns at 0.0 "$ftp start"
$ns at 0.0 "$cbr start"
$ns at 5.0 "finish"
$ns run
```

Output:



Result:

Thus the simulation of distance vector routing protocol is done successfully.

EX.NO.9B

SIMULATION OF LINK STATE ROUTING ALGORITHM

Aim:

To simulate the Distance vector and Link state routing protocols using NS2.

Algorithm:

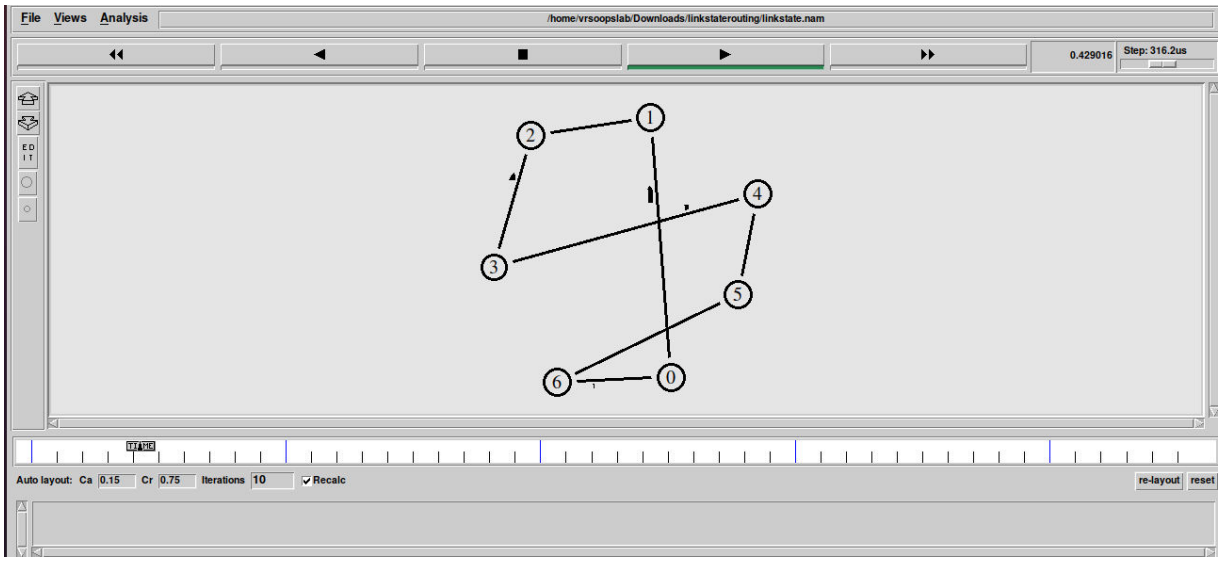
1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.
7. Create the applications and attach them to the UDP/TCP agent.
8. Connect udp/tcp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

Program:

```
set ns [new Simulator]
$ns rtproto LS
set nf [open linkstate.nam w]
$ns namtrace-all $nf
set f0 [open linkstate.tr w]
$ns trace-all $f0
proc finish {} {
global ns f0 nf
$ns flush-trace
close $f0
close $nf
exec nam linkstate.nam &
exit 0
}
for {set i 0} {$i <7} {incr i} {
set n($i) [$ns node]
}
for {set i 0} {$i <7} {incr i} {
$ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb 10ms DropTail
}
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0
```

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

Output:



Result:

Thus the simulation of link state routing protocol is done successfully.

EX.NO.10

SIMULATION OF ERROR CORRECTION CODE (LIKE CRC)

Aim :

To write a program in Java to implement the Simulation of Error Correction Code (CRC)

Algorithm :

At sender side

1. Start the program
2. Read the number of bits to be sent. Let n be the Number of bits in data to be sent from sender side.
3. Read the number of bits in the divisor. Let k be the Number of bits in the divisor (key obtained from generator polynomial).
4. The binary data is first increased by adding k-1 zeros in the end of the data
5. Use modulo-2 binary division to divide binary data by the divisor and store remainder of division.
6. Append the remainder at the end of the data to form the encoded data and send the same

At receiver side

1. Perform modulo-2 division again and if remainder is 0, then there are no errors.

Modulo 2 division

- In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend.
- The result of the XOR operation (remainder) is (n-1) bits, which is used for the next
- step after 1 extra bit is pulled down to make it n bits long.
- When there are no bits left to pull down, we have a result. The (n-1)-bit remainder
- which is appended at the sender side.

Program

```
import java.io.*;
import java.util.*;
class crc
{
 public static void main(String args[])
 {
        Scanner input = new Scanner(System.in);
        int n;
        // Read input
        System.out.println("Enter the size of the data:");
        n = input.nextInt();
        int data[] = new int[n];
        System.out.println("Enter the data, bit by bit:");
        for(int i=0 ; i < n ; i++)
        {
        System.out.println("Enter bit number " + (n-i) + ":");
```

```
        data[i] = input.nextInt();
        }
 //Read Divisor
        System.out.println("Enter the size of the divisor:");
        n = input.nextInt();
        int divisor[] = new int[n];
        System.out.println("Enter the divisor, bit by bit:");
        for(int i=0 ; i < n ; i++)
        {
        System.out.println("Enter bit number " + (n-i) + ":");
        divisor[i] = input.nextInt();
        }
//Perform Division
        int remainder[] = divide(data, divisor);
        for(int i=0 ; i < remainder.length-1 ; i++)
        {
        System.out.print(remainder[i]);
        }
        System.out.println("\nThe CRC code generated is:");
        for(int i=0 ; i < data.length ; i++)
        {
        System.out.print(data[i]);
        }
        for(int i=0 ; i < remainder.length-1 ; i++)
        {
        System.out.print(remainder[i]);
        }
        System.out.println();
// Append the remainder
        int sent_data[] = new int[data.length + remainder.length - 1];
        System.out.println("Enter the data to be sent:");
        for(int i=0 ; i < sent_data.length ; i++)
        {
        System.out.println("Enter bit number " + (sent_data.length-i)+ ":");
        sent_data[i] = input.nextInt();
        }
        receive(sent_data, divisor);
        }
        static int[] divide(int old_data[], int divisor[])
        {
        int remainder[] , i;
        int data[] = new int[old_data.length + divisor.length];
        System.arraycopy(old_data, 0, data, 0, old_data.length);
 // Remainder array stores the remainder
        remainder = new int[divisor.length];
```

```
// Initially, remainder's bits will be set to the data bits
        System.arraycopy(data, 0, remainder, 0, divisor.length);
        for(i=0 ; i < old_data.length ; i++)
        {
        System.out.println((i+1) + ".) First data bit is : " + remainder[0]);
        System.out.print("Remainder : ");
// If first bit of remainder is 1 then exor the remainder bits with divisor bits
        if(remainder[0] == 1)
        {
        for(int j=1 ; j < divisor.length ; j++)
        {
        remainder[j-1] = exor(remainder[j], divisor[j]);
        System.out.print(remainder[j-1]);
         }
        }
         else
        {
// If first bit of remainder is 0 then exor the remainder bits with 0
        for(int j=1 ; j < divisor.length ; j++)
        {
        remainder[j-1] = exor(remainder[j], 0);
        System.out.print(remainder[j-1]);
        }
         }
        remainder[divisor.length-1] = data[i+divisor.length];
        System.out.println(remainder[divisor.length-1]);
        }
        return remainder;
        }
        static int exor(int a, int b)
        {
        if(a == b)
        {
        return 0;
        }
        return 1;
        }
        static void receive(int data[], int divisor[])
        {
        int remainder[] = divide(data, divisor);
        for(int i=0 ; i < remainder.length ; i++)
        {
        if(remainder[i] != 0)
        {
```

```
        System.out.println("There is an error in received data...");
        return;
        }
        }
        System.out.println("Data was received without any error.");
        }
}
```

Output:

```
C:\Program Files\Java\jdk-19\bin>javac crc.java
C:\Program Files\Java\jdk-19\bin>java crc
Enter the size of the data:
7
Enter the data, bit by bit:
Enter bit number 7:
1
Enter bit number 6:
1
Enter bit number 5:
1
Enter bit number 4:
1
Enter bit number 3:
1
Enter bit number 2:
1
Enter bit number 1:
1
Enter the size of the divisor:
1
Enter the divisor, bit by bit:
Enter bit number 1:
1
1.) First data bit is : 1
Remainder : 1
2.) First data bit is : 1
Remainder : 1
3.) First data bit is : 1
Remainder : 1
4.) First data bit is : 1
Remainder : 1
5.) First data bit is : 1
Remainder : 1
6.) First data bit is : 1
Remainder : 1
```

7.) First data bit is : 1
Remainder : 0

The CRC code generated is:
1111111
Enter the data to be sent:
Enter bit number 7:
1
Enter bit number 6:
11
Enter bit number 5:
1
Enter bit number 4:
1
Enter bit number 3:
1
Enter bit number 2:
1
Enter bit number 1:
1
1.) First data bit is : 1
Remainder : 11
2.) First data bit is : 11
Remainder : 1
3.) First data bit is : 1
Remainder : 1
4.) First data bit is : 1
Remainder : 1
5.) First data bit is : 1
Remainder : 1
6.) First data bit is : 1
Remainder : 1
7.) First data bit is : 1
Remainder : 0
Data was received without any error.

C:\Program Files\Java\jdk-19\bin>1
'1' is not recognized as an internal or external command,
operable program or batch file.

C:\Program Files\Java\jdk-19\bin>java crc
Enter the size of the data:
7
Enter the data, bit by bit:
Enter bit number 7:

```
1
Enter bit number 6:
0
Enter bit number 5:
0
Enter bit number 4:
1
Enter bit number 3:
1
Enter bit number 2:
0
Enter bit number 1:
1
Enter the size of the divisor:
4
Enter the divisor, bit by bit:
Enter bit number 4:
1
Enter bit number 3:
0
Enter bit number 2:
1
Enter bit number 1:
1
1.) First data bit is : 1
Remainder :  0101
2.) First data bit is : 0
Remainder :  1010
3.) First data bit is : 1
Remainder :  0011
4.) First data bit is : 0
Remainder :  0110
5.) First data bit is : 0
Remainder :  1100
6.) First data bit is : 1
Remainder :  1110
7.) First data bit is : 1
Remainder : 1010
101
The CRC code generated is:
1001101101
Enter the data to be sent:
Enter bit number 10:
1
Enter bit number 9:
```

```
0
Enter bit number 8:
0
Enter bit number 7:
1
Enter bit number 6:
1
Enter bit number 5:
0
Enter bit number 4:
1
Enter bit number 3:
1
Enter bit number 2:
0
Enter bit number 1:
1
1.) First data bit is : 1
Remainder :  0101
2.) First data bit is : 0
Remainder :  1010
3.) First data bit is : 1
Remainder :  0011
4.) First data bit is : 0
Remainder :  0111
5.) First data bit is : 0
Remainder :  1110
6.) First data bit is : 1
Remainder :  1011
7.) First data bit is : 1
Remainder :  0000
8.) First data bit is : 0
Remainder :  0000
9.) First data bit is : 0
Remainder : 0000
10.) First data bit is : 0
Remainder : 0000
Data was received without any error.
```

Result:

Thus a program in Java implemented the Simulation of Error Correction Code (CRC).

ADDITTIONAL EXERIMENTS

EX.NO: 11

PERFORMANCE COMPARISONS OF MAC PROTOCOLS

Aim:

To compare various MAC Protocols performance using NS-2

Algorithm:

Step 1: Start network simulator OTCL editor.
Step 2: Create new simulator using set ns [new Simulator] syntax
Step 3: Create Trace route to Network Animator
set nf [open out.nam w]
$ns namtrace-all $nf
Step 4: Create procedure to trace all path
proc finish {} {
 global ns nf
 $ns flush-trace
 #Close the NAM trace file
 close $nf
 #Execute NAM on the trace file
 exec nam out.nam &
 exit 0
}
Step 4: Connect with TCP and SINK command.
$ns connect $tcp $sink
Step 5: Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
Step 6: Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
Step 7: Run and Execute the program.
$ns run

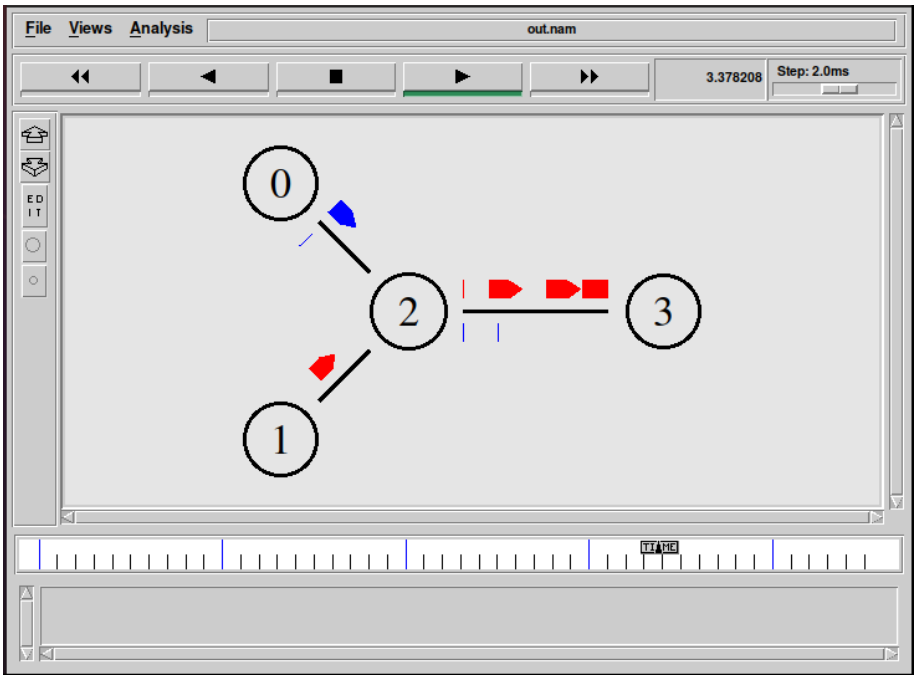
Program:

```
#Create a simulator object
set ns [new Simulator]
#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
#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 NAM trace file
 close $nf
 #Execute NAM on the trace file
 exec nam out.nam &
 exit 0
}
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail
#Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10
#Give node position (for NAM)
$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 link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5
#Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $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
#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $udp $null
```

```
$udp set fid_ 2
#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false
#Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
#Detach tcp and sink agents (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"
#Run the simulation
$ns run
```

Output:



Result:

Thus the MAC Protocols performance compared by using NS-2 and output verified by using Network Animator.