SCAD COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BE- Computer Science and Engineering

Anna University Regulation: 2021

CS3591- Computer Networks

III Year/V Semester

LAB MANUAL

Prepared By,

Mr. S.RAM PRASATH, AP/CSE

Ex.No.1 Learn to use commands like tcpdump, netstat, ifconfig, nslookup and trace route. Capture ping and trace route PDUs using a network protocol analyzer and examine

Aim:

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

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

$ su

(and password 123456)
$ sudo –i to change #
($ 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@localhost cse]# tcpdump -D
1.nflog (Linux netfilter log (NFLOG) interface)
2.nfqueue (Linux netfilter queue (NFQUEUE) interface)
3.usbmon1 (USB bus number 1)
4. enp2s0
5. usbmon2 (USB bus number 2)
6. any (Pseudo-device that captures on all interfaces)
7.lo [Loopback]

Capture all packets in any interface by running this command:

[root@localhost cse]# tcpdump -i any

06:03:58.258143 ARP, Request who-has 172.16.51.87 tell 172.16.22.25, length 46
06:03:58.258225 ARP, Request who-has 172.16.51.88 tell 172.16.22.25, length 46

06:03:58.260828 ARP, Request who-has 172.16.51.122 tell 172.16.22.25, length 46
06:03:58.260903 ARP, Request who-has 172.16.51.123 tell 172.16.22.25, length 46
^C
5244 packets captured
59636 packets received by filter
54378 packets dropped by kernel
(Press ctrl+C to stop execution)

Filter packets based on the source or destination IP Address

[root@localhost cse]#tcpdump -i any -c5 -nn src 172.16.20.138

6:10:30.712414 ARP, Request who-has 172.16.16.16 tell 172.16.20.138, length 28

06:10:31.483765 IP 172.16.20.138.47997 > 51.158.186.98.123: NTPv4, Client, length 48

5 packets captured

5 packets received by filter

0 packets dropped by kernel

[root@localhost cse]#tcpdump -i any -c5 -nn dst 172.16.20.139

6:10:30.712414 ARP, Request who-has 172.16.16.16 tell 172.16.20.138, length 28

06:10:31.483765 IP 172.16.20.138.47997 > 51.158.186.98.123: NTPv4, Client, length 48

5 packets captured

5 packets received by filter

0 packets dropped by kernel

Filtering packets

To filter packets based on protocol, specifying the protocol in the command line. For example, capture ICMP packets only by using this command:

[root@localhost cse]# tcpdump -i any -c5 icmp

(tcpdump captures and displays only the ICMP-related packets.)

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
06:15:07.800786 IP localhost.localdomain > ec2-54-204-39-132.compute-1.amazonaws.com: ICMP echo request, id 8180, seq 13, length 64
06:15:08.063488 IP ec2-54-204-39-132.compute-1.amazonaws.com > localhost.localdomain: ICMP echo reply, id 8180, seq 13, length 64

5 packets captured
5 packets received by filter
0 packets dropped by kernel

In a different terminal, try to ping another machine:
$ ping opensource.com

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.

[root@localhost cse]# netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp 0 0 localhost.localdo:53318 ec2-52-206-98-166:https ESTABLISHED
tcp 0 0 localhost.localdo:36418 sg2plpkivs-v03.any:http TIME_WAIT

-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

3. ifconfig

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

[cse@localhost ~]$ ifconfig

enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 172.16.20.138 netmask 255.255.0.0 broadcast 172.16.255.255
inet6 fe80::d884:13bc:fd22:2d43 prefixlen 64 scopeid 0x20<link>
ether a0:8c:fd:e7:10:86 txqueuelen 1000 (Ethernet)
RX packets 4474083 bytes 280780119 (267.7 MiB)
RX errors 0 dropped 353 overruns 0 frame 0
TX packets 14455 bytes 1798944 (1.7 MiB)
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 4154 bytes 352264 (344.0 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 4154 bytes 352264 (344.0 KiB)
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.

[cse@localhost ~]$ nslookup annauniv.edu
Server: 8.8.8.8
Address: 8.8 8.8#53

Non-authoritative answer:
Name: annauniv.edu
Address: 103.70.60.38

[cse@localhost ~]$ nslookup 172.217.26.206
Server: 8.8.8.8
Address: 8.8 8.8#53

Non-authoritative answer:
206.26.217.172.in-addr.arpa name = maa03s23-in-f14.1e100.net.
206.26.217.172.in-addr.arpa name = maa03s23-in-f14.1e100.net.
206.26.217.172.in-addr.arpa name = maa03s23-in-f206.1e100.net.
206.26.217.172.in-addr.arpa name = maa03s23-in-f206.1e100.net.

Authoritative tracerouteanswers can be found from:

Lookup for any record
[cse@localhost ~]$ nslookup -type=any annauniv.edu
Server: 8.8.8.8
Address: 8.8 8.8#53

Non-authoritative answer:
Name: annauniv.edu
Address: 103.70.60.38
annauniv.edu text = "v=spf1 ip4:103.70.60.40 -all"
annauniv.edu mail exchanger = 0 sonic.annauniv.edu.
annauniv.edu
origin = ns.annauniv.edu
mail addr = root.annauniv.edu
serial = 20170907
refresh = 300

retry = 900
expire = 604800
minimum = 86400
annauniv.edu nameserver = ns.annauniv.edu.

Authoritative answers can be found from:

Lookup for an ns record

[cse@localhost ~]$ nslookup -type=ns annauniv.edu
Server: 8.8.8.8
Address: 8.8 8.8#53

Non-authoritative answer:
annauniv.edu nameserver = ns.annauniv.edu.

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.

[cse@localhost ~]$ traceroute

Usage:
traceroute [-46dFITnreAUDV] [-f first_ttl] [-g gate, ...] [-i device] [-m max_ttl] [-N squeries] [-p port] [-t tos] [-l flow_label] [-w waittime] [-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

[cse@localhost ~]$ traceroute annauniv.edu
traceroute to annauniv.edu (103.70.60.38), 30 hops max, 60 byte packets
1 117.193.124.33 (117.193.124.33) 1.389 ms 1.216 ms 1.072 ms
2 172.16.199.74 (172.16.199.74) 1.902 ms 1.834 ms 1.761 ms
3 218.248.235.161 (218.248.235.161) 27.212 ms * *
4 * * *
5 218.248.178.42 (218.248.178.42) 15.521 ms * *
6 * * *
7 madurai-eg-175.232.249.45.powergrid.in (45.249.232.175) 16.007 ms 15.345 ms 15.867 ms

[cse@localhost ~]$ traceroute 172.16.20.139
traceroute to 172.16.20.139 (172.16.20.139), 30 hops max, 60 byte packets

1 localhost.localdomain (172.16.20.138) 3004.348 ms !H 3004.215 ms !H 3004.104 ms !H

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

Network protocol analyzer - 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

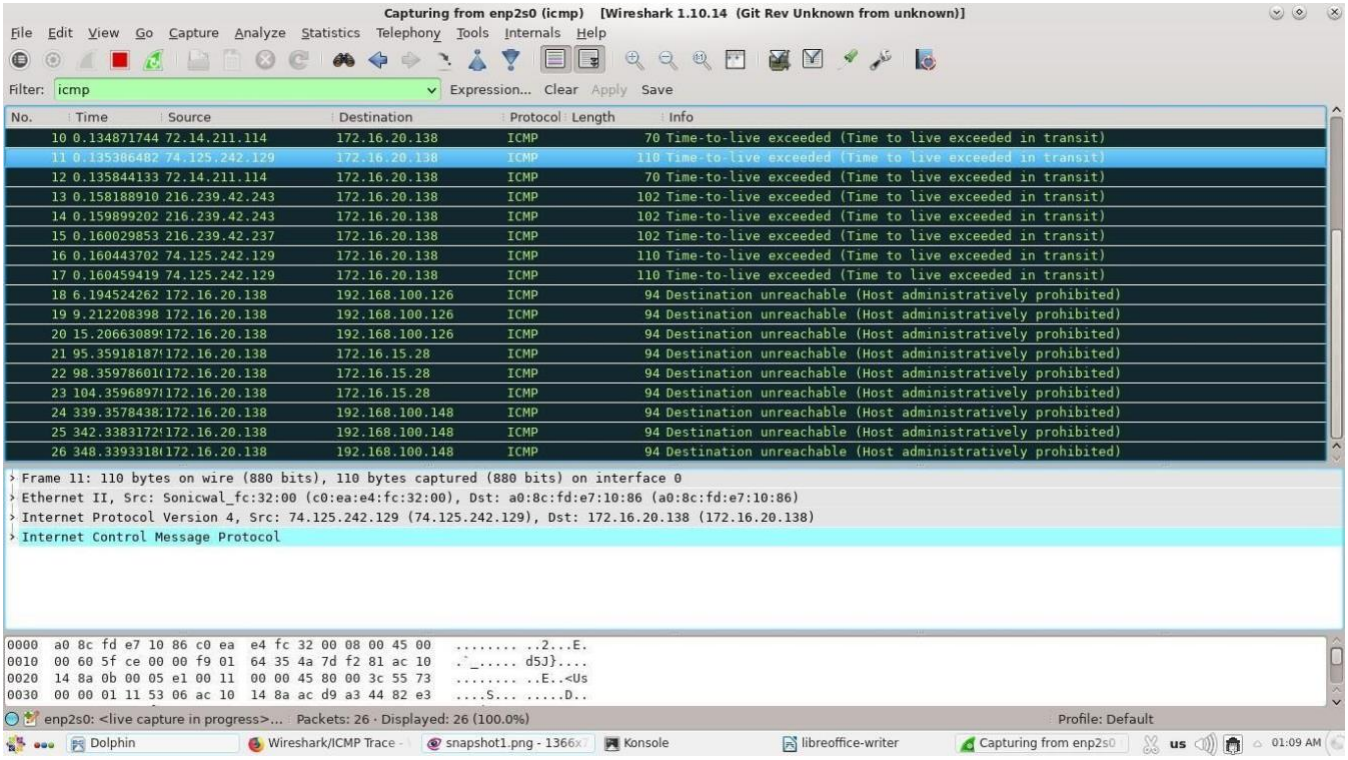
To Open Wireshark

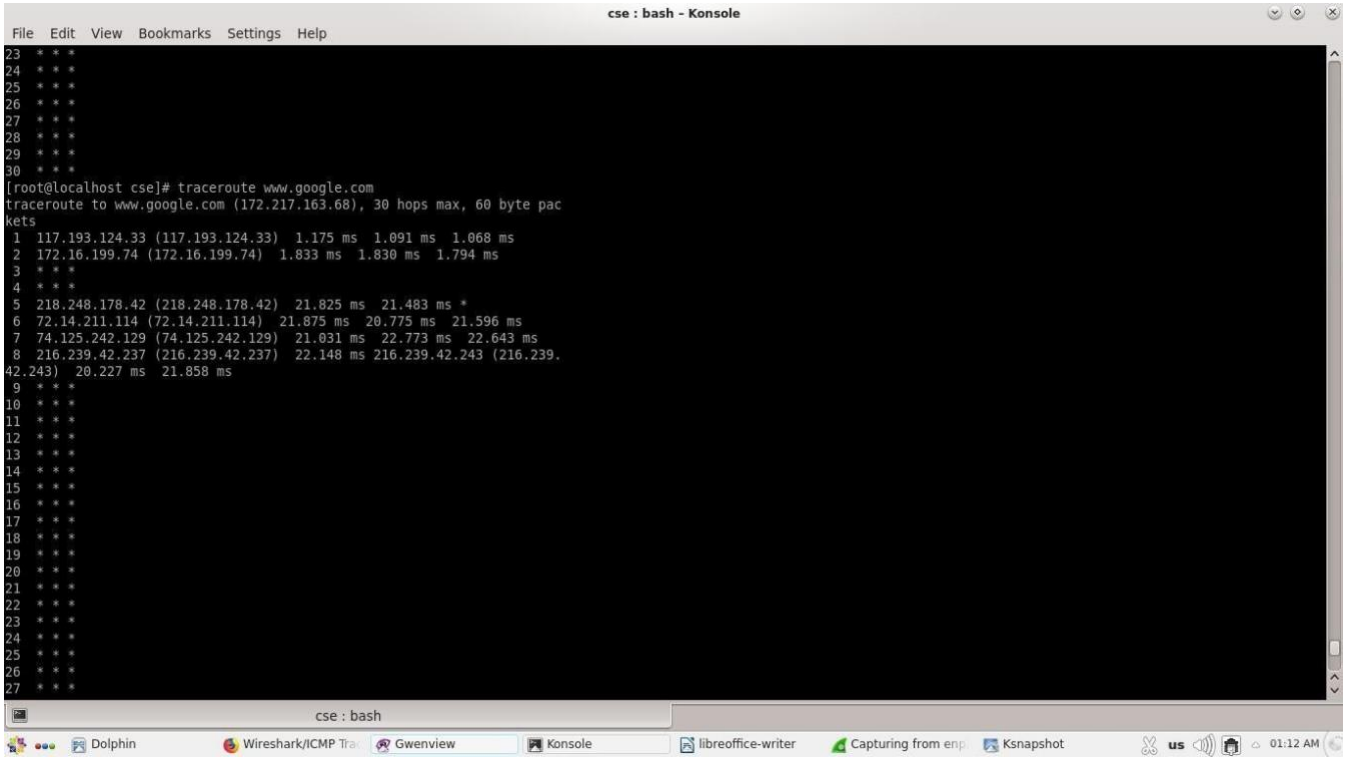
Open directly or use the following commands

 # sudo wireshark

In wireshark filter icmp packets

In a konsole execute
ping www.sudo.com
traceroute www.google.com





Result:

Thus commands like tcpdump, netstat, ifconfig, nslookup and traceroute was used. Ping and traceroute PDUs using a network protocol analyzer was captured and examined.

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

Aim:

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

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

Download.java

```
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

```
F:\Abarna\Lab>javac Download.java
F:\ Abarna \Lab>java Download
Downloading File From: http://tutorialspoint.com/java_dip/images/digital_image_processing.jpg
Buffer Read of length: 2048
Buffer Read of length: 2048
Buffer Read of length: 2048
Buffer Read of length: 2048
Buffer Read of length: 2048
Buffer Read of length: 1097
Buffer Read of length: 2048
Buffer Read of length: 2048
Buffer Read of length: 1744
Buffer Read of length: 2048
Buffer Read of length: 2048
Buffer Read of length: 2048
Buffer Read of length: 2048
Buffer Read of length: 2048
Buffer Read of length: 1440
Buffer Read of length: 2048
Buffer Read of length: 2048
```

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 548

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 2048

Buffer Read of length: 1863

The downloaded file (Stored in the current working directory)

Result:

Thus created a program in java for a HTTP web client program to download a web page using TCP sockets is written and executed successfully.

Ex.No.3a APPLICATION USING TCP SOCKETS - ECHO CLIENT AND ECHO SERVER

Aim:

To write a program in Java to implement an applications using TCP Sockets like echo client and echo server

Algorithm

1. Start the Program
2. In Server
    a) Create a server socket and bind it to port.
    b) Listen for new connection and when a connection arrives, accept it.
    c) Read the data from client.
    d) Echo the data back to the client.
    e) Close all streams.
    f) Close the server socket.
    g) Stop.
3. In Client
    a) Create a client socket and connect it to the server's port number.
    b) Send user data to the server.
    c) Display the data echoed by the server.
    d) Close the input and output streams.
    e) Close the client socket.
    f) Stop.
4. Stop the program

Program

Server.java

```
import java.net.*;
import java.lang.*;
import java.io.*;
public class Server
{
    public static final int PORT = 4000;
    public static void main( String args[])
```

```
{
        ServerSocket sersock = null;
        Socket sock = null;
        try
        {
                sersock = new ServerSocket(PORT);
                System.out.println("Server Started :"+sersock);
                try
                {
                        sock = sersock.accept();
                        System.out.println("Client Connected :"+ sock);
                DataInputStream ins = new DataInputStream(sock.getInputStream());
                        System.out.println(ins.readLine());
                        PrintStream ios = new PrintStream(sock.getOutputStream());
                        ios.println("Hello from server");
                        ios.close();
                        sock.close();
                }
                catch(SocketException se)
                {
                        System.out.println("Server Socket problem "+se.getMessage());
                }
        }
        catch(Exception e)
        {
                System.out.println("Couldn't start " + e.getMessage()) ;
        }
        System.out.println(" Connection from : " + sock.getInetAddress());
}
```

}

Client.java

```
import java.lang.*;
import java.io.*;
import java.net.*;
import java.net.InetAddress;
class client
{
	public static void main(String args[])
	{
		Socket sock=null;
		DataInputStream dis=null;
		PrintStream ps=null;
		System.out.println(" Trying to connect");
		try
		{
			sock= new Socket(InetAddress.getLocalHost(),Server.PORT);
			ps= new PrintStream(sock.getOutputStream());
			ps.println(" Hi from client");
			DataInputStream is = new DataInputStream(sock.getInputStream());
			System.out.println(is.readLine());
		}
		catch(SocketException e)
		{
			System.out.println("SocketException " + e);
		}
		catch(IOException e)
		{
			System.out.println("IOException " + e);
		}
		finally
```

```
            {
                try
                {
                    sock.close();
                }

                catch(IOException ie)
                {
                    System.out.println(" Close Error :" + ie.getMessage());
                }
            }

        }
}
```

Output

In server window:

F:\ Abarna \Lab>javac Server.java

F:\ Abarna \Lab>java Server

Server Started :ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=4000]

Client Connected :Socket[addr=/192.168.1.18,port=1815,localport=4000]

Hi from client

Connection from : /192.168.1.18

In client window:

F:\ Abarna\Lab>javac client.java

F:\ Abarna\Lab>java client

Trying to connect

Hello from server

Result:

Thus a program in Java implemented an applications using TCP Sockets like echo client and echo server

Ex.No.3b **APPLICATION USING TCP SOCKETS - CHAT**

Aim

To write a program in Java to implement an applications using TCP Sockets like chat.

Algorithm

1. Start the Program
2. In Server
   a) Create a server socket and bind it to port.
   b) Listen for new connection and when a connection arrives, accept it.
   c) Read Client's message and display it
   d) Get a message from user and send it to client
   e) Repeat steps 3-4 until the client sends "end"
   f) Close all streams
   g) Close the server and client socket
   h) Stop
3. In Client
   a) Create a client socket and connect it to the server's port number
   b) Get a message from user and send it to server
   c) Read server's response and display it
   d) Repeat steps 2-3 until chat is terminated with "end" message
   e) Close all input/output streams
   f) Close the client socket
   g) Stop
4. Stop the program

Program

tcpchatserver.java

```
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(4000);
        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("Server          : " +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());
                }
        }
}
```

tcpchatclient.java

```
import java.io.*;
import java.net.*;
class tcpchatclient
{
        public static void main(String args[])throws Exception
        {
                Socket Clt;
                PrintWriter toServer;
                BufferedReader fromUser, fromServer;
                try
                {
                    Clt = new Socket(InetAddress.getLocalHost(),4000);
                    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("Message 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

In server window:

F:\ Abarna\ Lab>javac tcpchatserver.java

F:\ Abarna\Lab>java tcpchatserver

Server started

Client connectedServer : hai

Message to Client : hello

Client Disconnected

In client window:

F:\ Abarna\Lab>javac tcpchatclient.java

F:\ Abarna\Lab>java tcpchatclient

Type "end" to Quit

Message to Server : hai

Client : hello

Message to Server : end

Result:

Thus a program in Java implemented an application using TCP Sockets like chat.

Ex.No.4 **SIMULATION OF DNS USING UDP SOCKETS**

Aim :

To write a program in Java to perform Simulation of DNS using UDP sockets.

Algorithm

1. Start the Program
2. In Server
   a) Create an array of hosts and its ip address in another array
   b) Create a datagram socket and bind it to a port
   c) Create a datagram packet to receive client request
   d) Read the domain name from client to be resolved
   e) Lookup the host array for the domain name
   f) If found then retrieve corresponding address
   g) Create a datagram packet and send ip address to client
   h) Repeat steps 3-7 to resolve further requests from clients
   i) Close the server socket
   j) Stop
3. In Client
   a) Create a datagram socket
   b) Get domain name from user
   c) Create a datagram packet and send domain name to the server
   d) Create a datagram packet to receive server message
   e) Read server's response
   f) If ip address then display it else display "Domain does not exist"
   g) Close the client socket
   h) Stop
4. Stop the program

Program

dnsclient.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 = 8080;
                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();
        }
}
```

dnsserver.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 = {"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(8080);
                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

In server window

F:\ Abarna\Lab>javac dnsserver.java

F:\ Abarna\Lab>java dnsserver

Press Ctrl + C to Quit

Request for host yahoo.com

In client window

F:\ Abarna\Lab>javac dnsclient.java

F:\ Abarna\Lab>java dnsclient

Enter the hostname : yahoo.com

IP Address: 68.180.206.184

Result :

Thus a program in Java performed Simulation of DNS using UDP sockets.

Ex.No.5 USE A TOOL LIKE WIRESHARK TO CAPTURE PACKETS AND EXAMINE THE PACKETS

Aim :

To implement the code to capture packets and examine the packets using wireshark.

Procedure:

1. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.
2. Identifying and analyzing protocols.
3. Identifying source & destination of traffic.

Program :

```
import sys

from scapy.all import *

# Define the packet capturing

functiondef

packet_handler(packet):

  print(packet.show())

# Capture packets on the network

interfacesniff(iface='eth0',

prn=packet_handler)
```

Output:

```
###[ Ethernet ]###
dst   = 00:11:22:33:44:55
src   = 66:77:88:99:00:11
type  = IPv4 ###[ IP ]###
version = 4
ihl   = 5
tos   = 0x0
len   = 1500
id    = 52786
flags = DF
frag  = 0
ttl   = 128
proto= tcp
chksum= 0xe877
src   = 192.168.1.10
dst   = 216.58.194.78
```

\options \ ###[TCP]###
sport= 54321
dport= http
seq = 3665840854
ack = 2542707921
dataofs = 5
reserved = 0
flags = FA
window= 8760
chksum= 0x9d6c
urgptr= 0
options = [('MSS', 1460), ('SAckOK', ''), ('Timestamp', (12576855, 2413)), ('NOP', None), ('WScale', 7)] ###[Raw]###
load = ''

Result:

Thus the program was executed successfully using tool Wireshark to capture packet and examine the packet.

Ex.No.6a SIMULATION OF ARP PROTOCOLS

Aim:

To write a program in java to simulate ARP protocols.

Algorithm:

1. Start the program
2. In Client
    a. Start the program
    b. Using socket connection is established between client and server.
    c. Get the IP address to be converted into MAC address.
    d. Send this IP address to server.
    e. Server returns the MAC address to client.
3. In Server
    a. Start the program
    b. Accept the socket which is created by the client.
    c. Server maintains the table in which IP and corresponding MAC addresses are stored.
    d. Read the IP address which is send by the client.
    e. Map the IP address with its MAC address and return the MAC address to client.
4. Stop the program

Program

Clientarp.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientarp
{
public static void main(String args[])
{
try
{
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
Socket clsct=new Socket("127.0.0.1",139);
DataInputStream din=new DataInputStream(clsct.getInputStream());
DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
System.out.println("Enter the Logical address(IP):");
```

```
String  str1=in.readLine();
dout.writeBytes(str1+'\n');
String str=din.readLine();
System.out.println("The Physical Address is: "+str);
clsct.close();
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```

Serverarp.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverarp
{
public static void main(String args[])
{
try
{
ServerSocket obj=new ServerSocket(139);
Socket obj1=obj.accept();
while(true)
{
DataInputStream din=new DataInputStream(obj1.getInputStream());
DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());
String str=din.readLine();
String ip[]={"165.165.80.80","165.165.79.1"};
String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
```

```
for(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);
}
}
}
```

Output

```
F:\ Abarna\2020-2021\Odd\Network Lab>java Serverarp
F:\ Abarna\2020-2021\Odd\Network Lab>java Clientarp
Enter the Logical address (IP): 165.165.80.80
The Physical Address is: 6A:08: AA: C2
```

Result:

Thus a program in java simulated ARP protocols.

Ex.No.6b SIMULATION OF RARP PROTOCOLS

Aim:

To write a program in java to simulate RARP protocols.

Algorithm:

1. Start the program
2. In Client
    a. Start the program
    b. Using socket connection is established between client and server.
    c. Get the MAC address to be converted into IP address.
    d. Send this MAC address to server.
    e. Server returns the IP address to client.
3. In Server
    a. Start the program
    b. Accept the socket which is created by the client.
    c. Server maintains the table in which IP and corresponding MAC addresses are stored.
    d. Read the MAC address which is send by the client.
    e. Map the MAC address with its MAC address and return the IP address to client.
4. Stop the program

Program

clientrarp.java

```
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[1204];
```

```
        byte[] receiveByte = new byte[1024];
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter the Physical Address ");
        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 :" + s.trim());
        client.close(); }
    catch(Exception e) {
    System.out.println(e);
    } } }
```

serverrarp.java

```
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[1204];
                byte[] receiveByte = new byte[1204];
                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[] = {"10.0.3.186"};
            String mac[] = {"D4:3D:7E:12:A3:D9"};
               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);
   }
   }
}
```

Output:

F:\ Abarna\2020-2021\Odd\Network Lab>java serverrarp F:\ Abarna\2020-2021\Odd\Network Lab>java clientrarp Enter the Physical address : D4:3D:7E:12:A3:D9

The Logical Address is: 10.0.3.186

Result:

Thus a program in java simulated RARP protocols.

Ex.No. 7a **STUDY OF NETWORK SIMULATOR (NS)**

AIM:

To study Network Simulator in detail.

INTRODUCTION :

Network Simulator (Version 2), widely known as NS2, is simply an eventdriven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989. Ever since, several revolutions and revisions have marked the growing maturity of the tool, thanks to substantial contributions from the players in the field. Among these are the University of California and Cornell University who developed the REAL network simulator, 1 the foundation which NS is based on.

Since 1995 the Defense Advanced Research Projects Agency (DARPA) supported development of NS through the Virtual InterNetwork Testbed (VINT) project currently the National Science Foundation (NSF) has joined the ride in development. Last but not the least, the group of Researchers and developers in the community are constantly working to keep NS2 strong and versatile.

BASIC ARCHITECTURE :

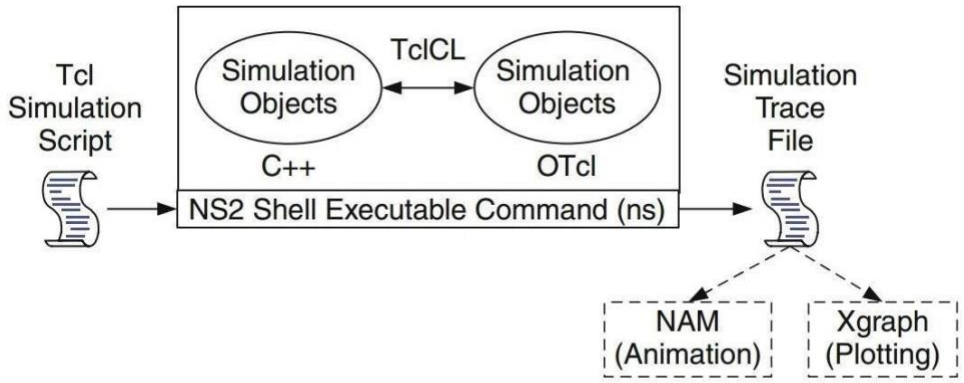


Fig. 2 Basic architecture of NS.

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

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

OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).

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

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

CONCEPT OVERVIEW

ns uses two languages because simulator has two different kinds of things it needs to do. On one hand, detailed simulations of protocols require a systems programming language which can efficiently manipulate bytes, packet headers and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring number of scenarios.

In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. ns meets both of these needs with two languages, C++ and OTcl.

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 ALGORITHMS USING NS

Aim:

To perform simulation of Congestion Control Algorithms (sliding window) using NS.

Algorithm

1. Run NSG 2.1
2. Create two nodes n0 and n1.
3. Create a duplex-link and set the following parameters
    a. Queuetype : Droptail
    b. Capacity 0.2 Mbps
    c. Propagation delay: 200 ms
    d. Queue size: 10
4. Create link from n0 to n1.
5. In Agent tab do the following
    a. Packet size: 500 bytes
    b. Agent type: TCP and draw a line from n0.
    c. Agent type: TCP Sink and draw a line from n1.

d. Draw a line from tcp to sink.

6. In Application tab do the following.
    a. Application Type: ftp
    b. Start time: 0.1
    c. Stop time: 3.5
    d. Draw line from tcp.
7. In parameters tab do the following
    a. Simulation time: 5.0
    b. Trace file: sliding.tr
    c. Nam file: sliding.nam
    d. Click done
8. Click TCL and specify the window side of the sliding window and the transfer of packets and save the file as sliding.tcl in C:\cygwin\Your folder

Program:

```
#=================================
#     Simulation parameters setup
#=================================
set val(stop)   5.0                ;# time of simulation end
```

```
#==================================
#        Initialization
#==================================
#Create a ns simulator
set ns [new Simulator]

#Open the NS trace file
set tracefile [open sliding.tr w]
$ns trace-all $tracefile

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

#==================================
#        Nodes Definition
#==================================
#Create 2 nodes

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

$ns at 0.0 "$n0 label Sender"
$ns at 0.0 "$n1 label Receiver"

#==================================
#        Links Definition
#==================================
#Createlinks between nodes

$ns duplex-link $n0 $n1 0.2Mb 200ms DropTail
$ns queue-limit $n0 $n1 10
```

```
#Give node position (for NAM)

$ns duplex-link-op $n0 $n1 orient right

#=================================
#        Agents Definition
#=================================
#Setup a TCP connection

set tcp1 [new Agent/TCP]

$tcp1 set windowInit_ 4
$tcp1 set maxcwnd_ 4

$ns attach-agent $n0 $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n1 $sink2
$ns connect $tcp1 $sink2
$tcp1 set packetSize_ 500

#=================================
#        Applications Definition
#=================================
#Setup a FTP Application over TCP connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp1

$ns add-agent-trace $tcp1 tcp
$ns monitor-agent-trace $tcp1
$tcp1 tracevar cwnd_

$ns at 0.1 "$ftp0 start"
$ns at 3.5 "$ftp0 stop"
```

$ns at 0.0 "$ns trace-annotate \"Sliding Window with window size 4 (normal operation)\""

$ns at 0.05 "$ns trace-annotate \"FTP starts at 0.1\""

$ns at 0.11 "$ns trace-annotate \"Send Packet 0,1,2,3\""

$ns at 0.34 "$ns trace-annotate \"Receive Ack 0,1,2,3\""

$ns at 0.56 "$ns trace-annotate \"Send Packet 4,5,6,7\""

$ns at 0.79 "$ns trace-annotate \"Receive Ack 4,5,6,7\""

$ns at 0.99 "$ns trace-annotate \"Send Packet 8,9,10,11\""

$ns at 1.23 "$ns trace-annotate \"Receive Ack 8,9,10,11 \""

$ns at 1.43 "$ns trace-annotate \"Send Packet 12,13,14,15\""

$ns at 1.67 "$ns trace-annotate \"Receive Ack 12,13,14,15\""

$ns at 1.88 "$ns trace-annotate \"Send Packet 16,17,18,19\""

$ns at 2.11 "$ns trace-annotate \"Receive Ack 16,17,18,19\""

$ns at 2.32 "$ns trace-annotate \"Send Packet 20,21,22,23\""

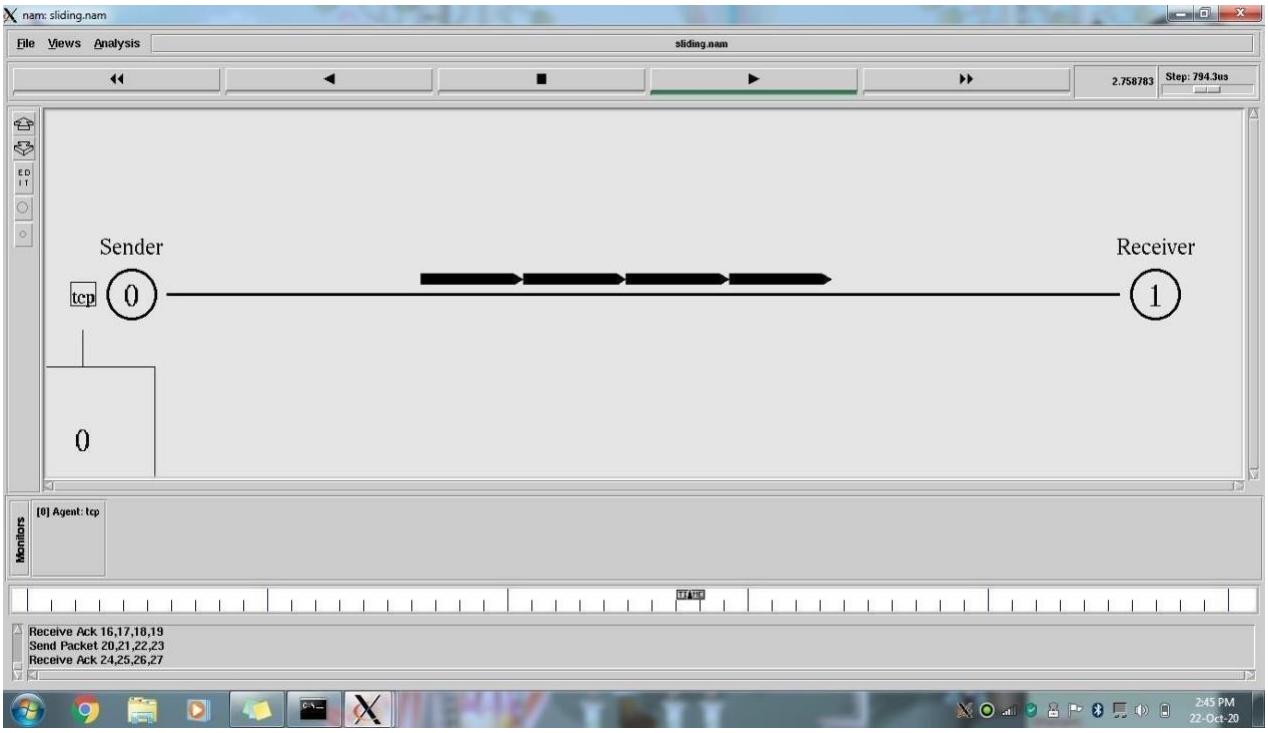
$ns at 2.56 "$ns trace-annotate \"Receive Ack 24,25,26,27\""

$ns at 2.76 "$ns trace-annotate \"Send Packet 28,29,30,31\""

$ns at 3.00 "$ns trace-annotate \"Receive Ack 28\""

$ns at 3.1 "$ns trace-annotate \"FTP stops\""

```
#===================================
#        Termination
#===================================
#Define a 'finish' procedure

proc finish {} {
    global ns tracefile namfile
    $ns  flush-trace
    close $tracefile
    close $namfile
    exec nam sliding.nam &
    exit 0
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run
```



Steps to execute:

1. Run cygwin and change the directory

a. Say for example

```
admin@CS03C037 ~
$  cd  ..
admin@CS03C037 /home
$ cd ..
Admin@CS03C037/
$ ls
Cygwin.bat Cygwin.ico Abarna bin cygdrive dev etc home lib opt proc tmp usr var
admin@CS03C037 /
$ cd Abarna
admin@CS03C037 /Abarna
$ ls
sliding.tcl
admin@CS03C037 /Abarna
$ ns sliding.tcl
admin@CS03C037 /Abarna
$ nam: no display name and no $DISPLAY environment variable
```

b. Run the tcl file as:

```
admin@CS03C037 /Abarna
$ ns sliding.tcl
```

c. Run the nam file:

i. Open nam-1.0a11a-win32.exe file.

ii. In File→Open select sliding.nam from C:\cygwin\Your folder

Result:

Thus performed simulation of Congestion Control Algorithms (sliding window) using NS.

Ex.No:8 STUDY OF TCP/UDP PERFORMANCE USING SIMULATION TOOL

Aim :

To study the performance of TCP/UDP using Simulation Tool (NS2)

Algorithm

1. Create 4 nodes (n0, n1, n2, n3).
2. The duplex links between n0 and n2, and n1 and n2 have 2 Mbps of bandwidth and 10 ms of delay.
3. The duplex link between n2 and n3 has 1.7 Mbps of bandwidth and 20 ms of delay.
4. Each node uses a DropTail queue, of which the maximum size is 10.
5. A "tcp" agent is attached to n0, and a connection is established to a tcp "sink" agent attached to n3.
6. As default, the maximum size of a packet that a "tcp" agent can generate is 1KByte.
7. A tcp "sink" agent generates and sends ACK packets to the sender (tcp agent) and frees the received packets.
8. A "udp" agent that is attached to n1 is connected to a "null" agent attached to n3.
9. A "null" agent just frees the packets received.
10. A "ftp" and a "cbr" traffic generator are attached to "tcp" and "udp" agents respectively, and the "cbr" is configured to generate 1 KByte packets at the rate of 1 Mbps.
11. The "cbr" is set to start at 0.1 sec and stop at 4.5 sec, and "ftp" is set to start at 1.0 sec and stop at 4.0 sec

Program

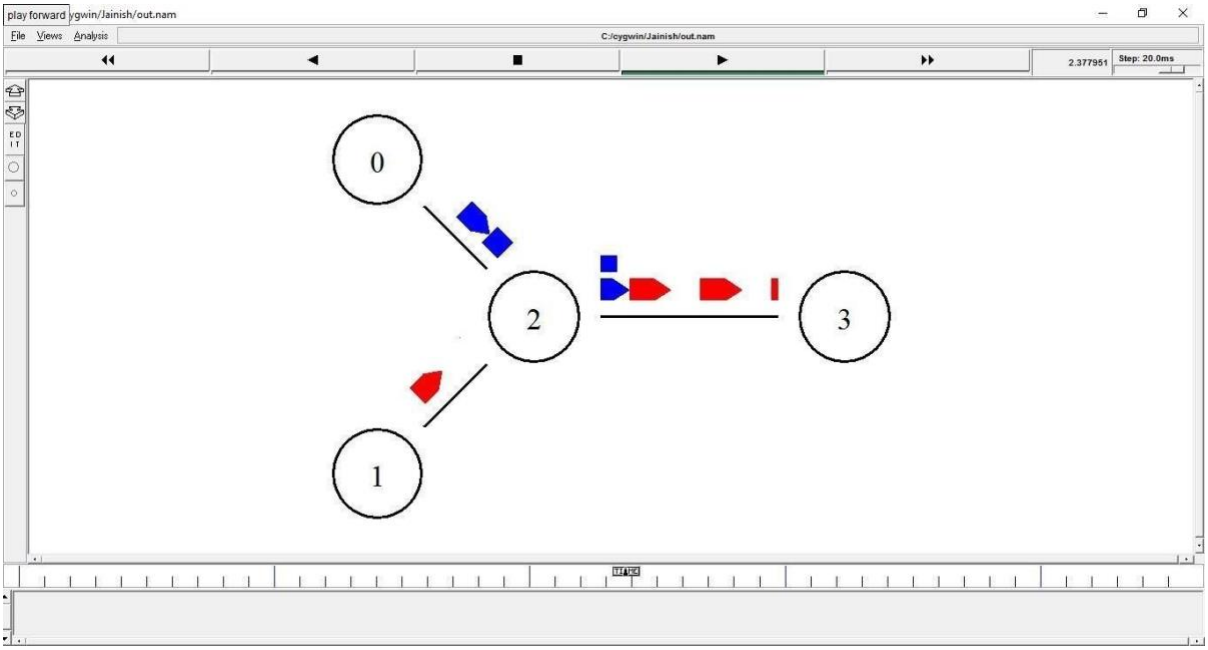
```
#Create a simulator object
set ns [new Simulator]

#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 studied the performance of TCP/UDP using Simulation Tool (NS2)

Ex.No.9a SIMULATION OF DISTANCE VECTOR ROUTING ALGORITHM

Aim:

To perform Simulation of Distance Vector Routing algorithm.

ALGORITHM:

1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose distance vector routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

Program (Distance Vector Protocol)

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

```
$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail

set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

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

set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1

set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0

$ns rtproto DV
```

$ns rtmodel-at 10.0 down $n(11) $n(5)

$ns rtmodel-at 15.0 down $n(7) $n(6)

$ns rtmodel-at 30.0 up $n(11) $n(5)

$ns rtmodel-at 20.0 up $n(7) $n(6)

$udp0 set fid_ 1

$udp1 set fid_ 2

$ns color 1 Red

$ns color 2 Green

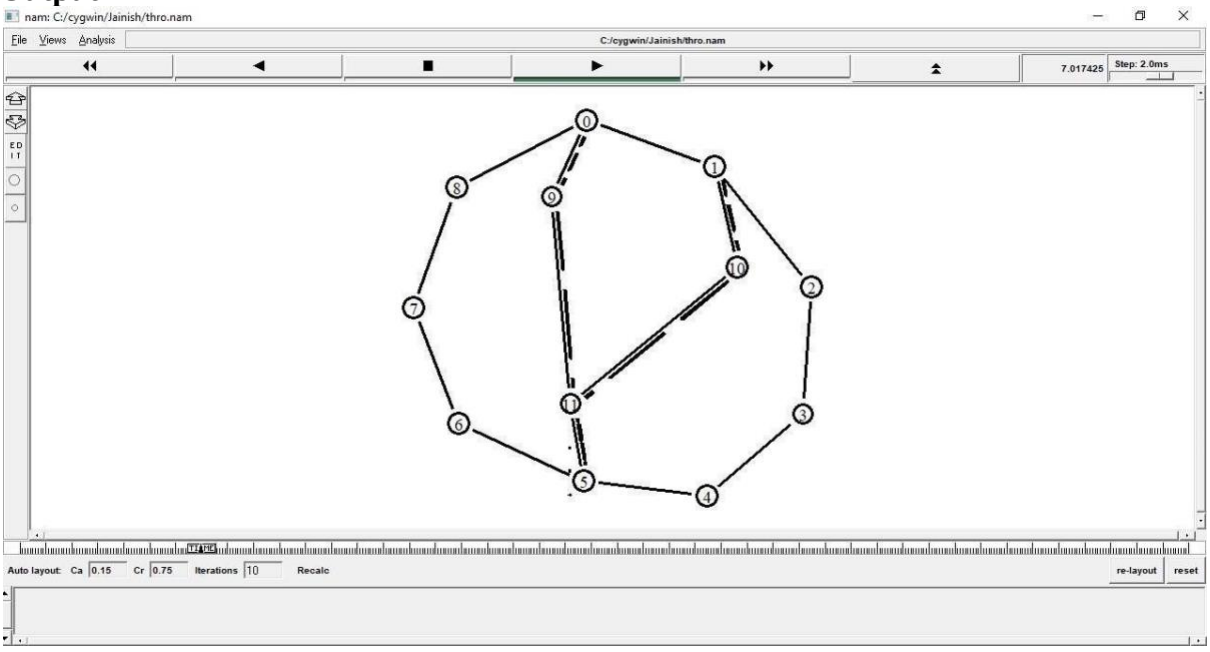
$ns at 1.0 "$cbr0 start"

$ns at 2.0 "$cbr1 start"

$ns at 45 "finish"

$ns run

Output



Result:

Thus performed Simulation of Distance Vector Routing algorithm.

Ex.No.9b SIMULATION OF LINK STATE ROUTING ALGORITHM

Aim:

To perform Simulation of Link State Routing algorithm.

Algorithm:

1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose Link state routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

Program (Link State Protocol)

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

```
$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail }

$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail

set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

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

set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1

set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0
$ns rtproto LS
```

$ns rtmodel-at 10.0 down $n(11) $n(5)

$ns rtmodel-at 15.0 down $n(7) $n(6)

$ns rtmodel-at 30.0 up $n(11) $n(5)

$ns rtmodel-at 20.0 up $n(7) $n(6)

$udp0 set fid_ 1

$udp1 set fid_ 2

$ns color 1 Red

$ns color 2 Green

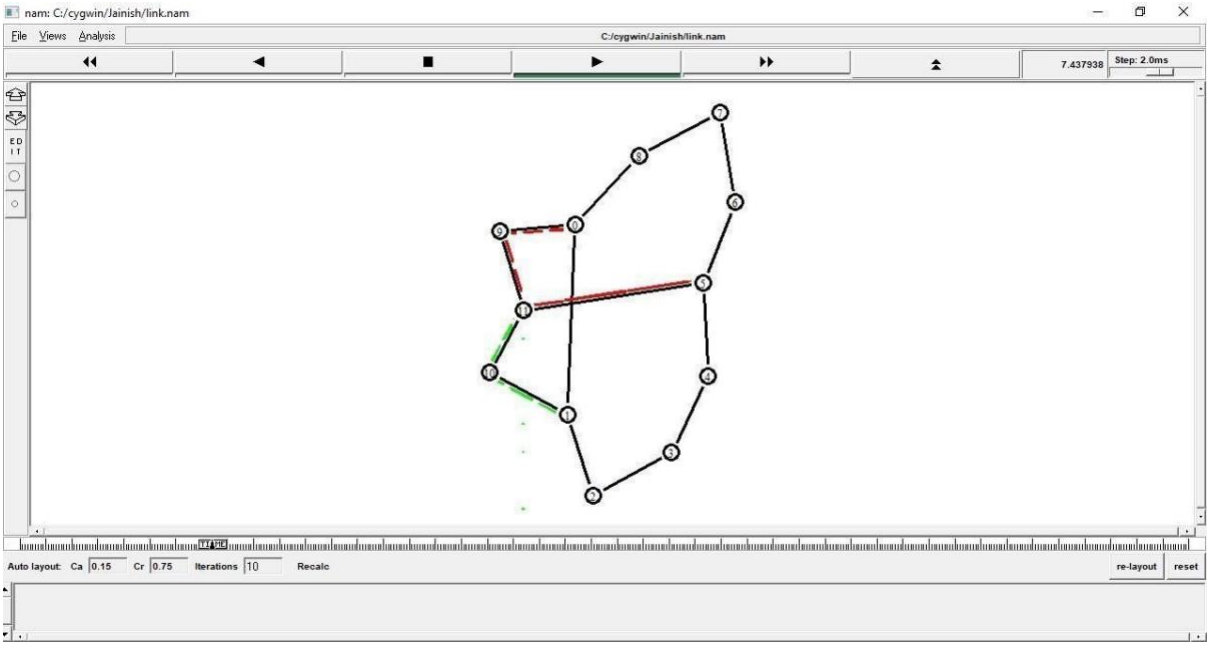
$ns at 1.0 "$cbr0 start"

$ns at 2.0 "$cbr1 start"

$ns at 45 "finish"

$ns run

Output:



Result:

Thus performed Simulation of Link State Routing algorithm.

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

```
D:\Abarna>javac crc.java
D:\Abarna>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 : 1Remainder :

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 : 0Remainder :

0000

Data was received without any error.

Result:

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