R.M.K. COLLEGE OF
ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)
(Sponsored by LAKSHMIKANNTHAMMAL EDUCATION TRUST)
Approved by All India Council for Technical Education, New Delhi and
Affiliated to Anna University, Chennai
All the Eligible Programs Accredited by NBA – NAAC with "A" Grade & ISO 9001 : 2005 Certified

R.S.M. Nagar, Puduvoyal - 601 206. Gummidipoondi Tk., Thiruvallur Dist. Tamilnadu, India.

Department : **COMPUTER SCIENCE AND ENGINEERING**

Laboratory : **22CS511 NETWORKS LABORATORY**

Semester : **V**

Certified that this is a bonafide record work done by ……..……

with Roll / Reg. Number .....He / She is a student

of ..... in

the **R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY, Puduvoyal.**

Faculty-in-Charge Head of the Department

Internal Examiner Date: External Examiner

INDEX

Ex. No.	Date	Name of the exercise	Page No.	Marks obtained	Signature of the faculty
1		Practice different network commands available in Windows and Linux Operating Systems and troubleshoot the network			
2		Network configuration commands using Linux			
3		Error detection and correction mechanisms			
4a		Flow control mechanisms: Implementation of Sliding Window Protocol			
4b		Flow control mechanisms: Implementation of Stop and Wait Protocol			
5		Multi-client chatting in TCP and UDP using Socket programming			
6		Implementation of HTTP, Web Caching, FTP using socket programming			
7		Develop a DNS client server to resolve the given host name or IP address			
8		Simulation of unicast routing protocols			
9		Observing Packets across the network and Performance Analysis of various Routing protocols			
10		Simulation of Transport layer Protocols and analysis of congestion control techniques in the network			

Ex. No: 01	Practice different network commands available in Windows and Linux Operating Systems and troubleshoot the network

Aim

To practice different network commands available in Windows and Linux Operating Systems like tcpdump, netstat, ifconfig, nslookup and traceroute ping.

Pre Lab-Discussion:

TCPDUMP
The tcpdump utility allows you to capture packets that flow within your network to assist in network troubleshooting. The following are several examples of using tcpdump with different options. Traffic is captured based on a specified filter.

NETSTAT
Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX, and other operating systems. Netstat provides information and statistics about protocols in use and current TCP/IP network connections.

IPCONFIG
ipconfig is a console application designed to run from the Windows command prompt. This utility allows you to get the IP address information of a Windows computer. From the command prompt, type ipconfig to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapter.

NSLOOKUP
The nslookup (which stands for name server lookup) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System.

TRACE ROUTE
Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Traceroute also records the time taken for each hop the packet makes during its route to the destination

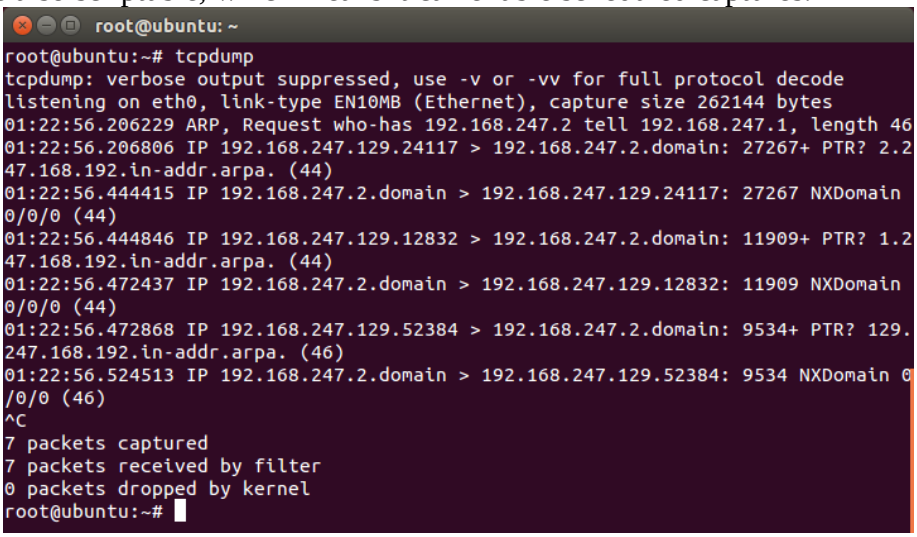
Commands:

1. tcpdump:

tcpdump is a data-network packet analyzer computer program that runs under a command line interface. It allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached.

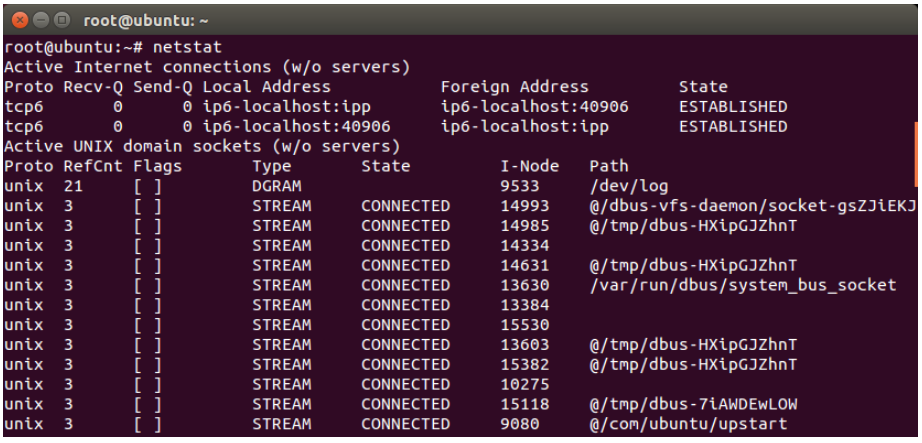
Tcpdump is a common open-source Linux tool used to analyse packets. It is fast, straightforward, and lightweight. Tcpdump is a simple application that works well in Linux servers without Linux-based network devices, a GUI or various IoT nodes. These attributes

give tcpdump an advantage over more powerful GUI-based analyzers, like Wireshark. Tcpdump is also scriptable, which means it can enable scheduled captures.



2. netstat

Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX, and other operating systems. Netstat provides information and statistics about protocols in use and current TCP/IP network connections. The Windows help screen analogous to a Linux or UNIX for netstat reads as follows:
displays protocol statistics and current TCP/IP network connections.
#netstat

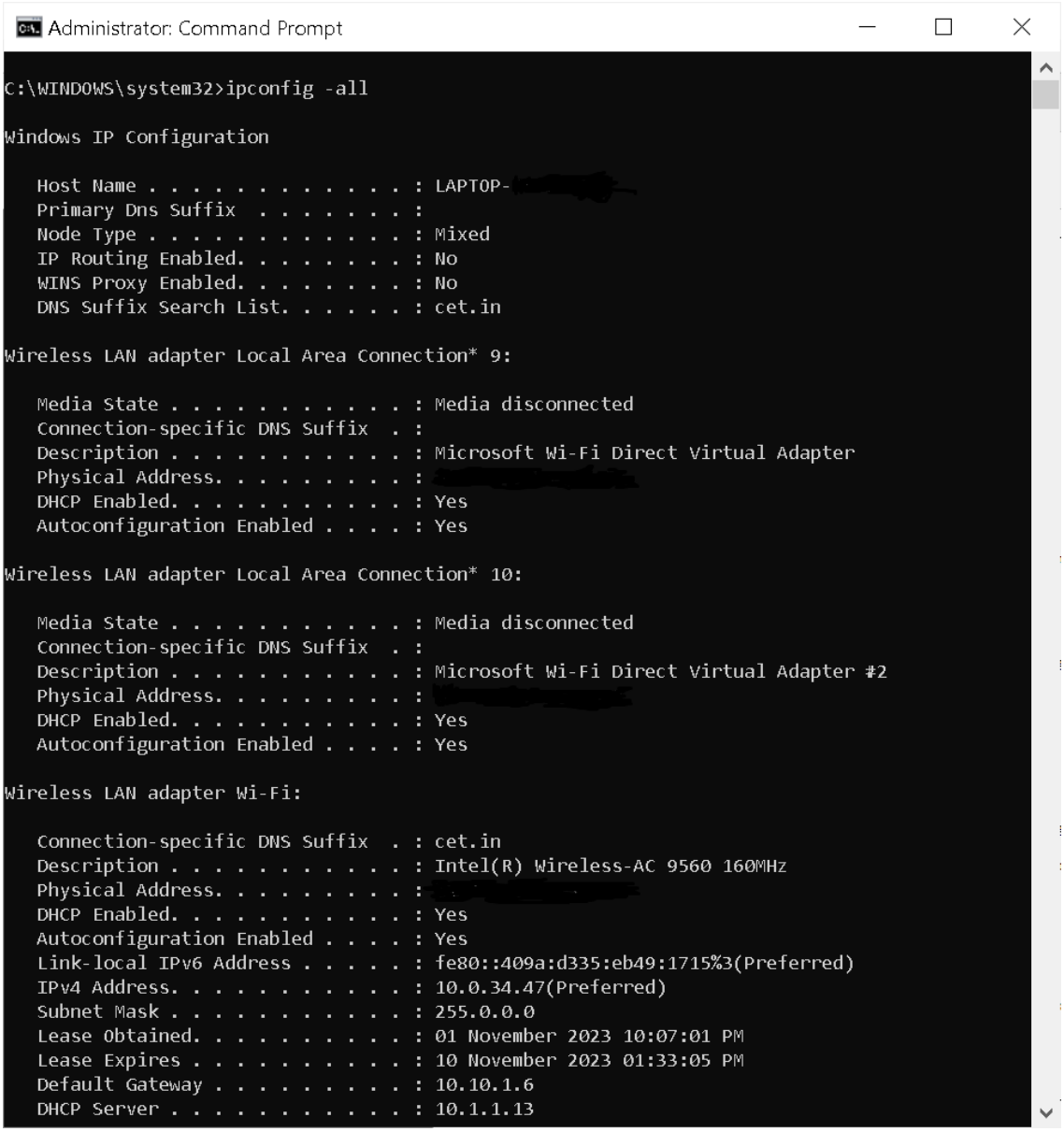


3. ipconfig

In Windows, ipconfig is a console application designed to run from the Windows command prompt. This utility allows you to get the IP address information of a Windows computer.

Using ipconfig

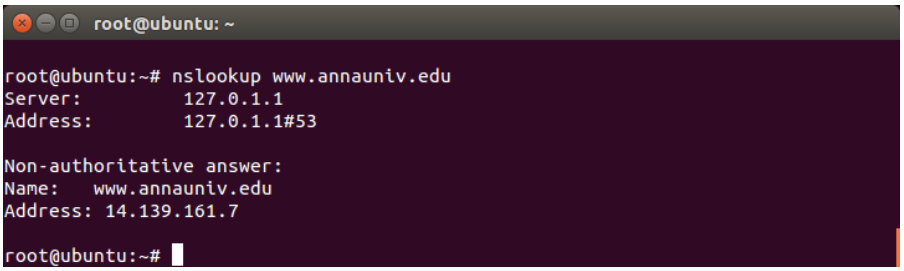
From the command prompt, type ipconfig to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapter.
> ipconfig



4. nslookup

The nslookup (which stands for name server lookup) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System. The nslookup command is a powerful tool for diagnosing DNS problems. You know you are experiencing a DNS problem when you can access a resource by specifying its IP address but not its DNS name.

#nslookup



5. Trace route:
Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values. The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that hop.

Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Traceroute also records the time taken for each hop the packet makes during its route to the destination. Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values.

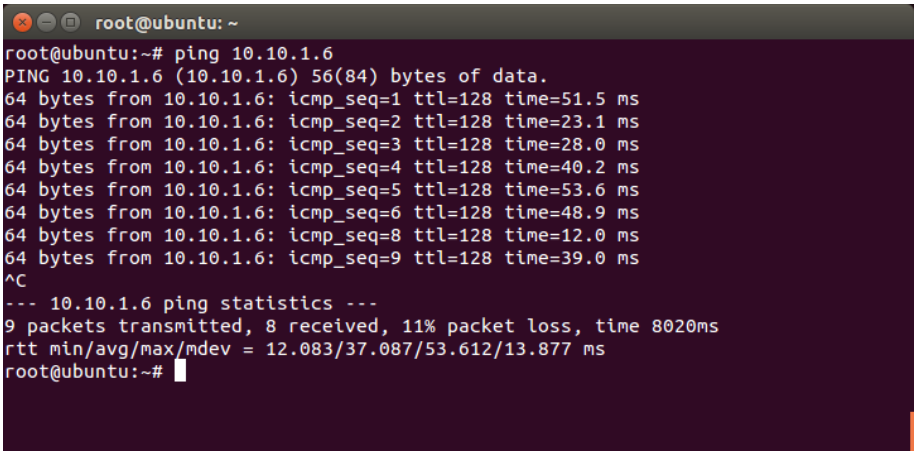
The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that hop. Traceroute sends packets with TTL values that gradually increase from packet to packet, starting with TTL value of one. Routers decrement TTL values of packets by one when routing and discard packets whose TTL value has reached zero, returning the ICMP error message ICMP Time Exceeded. For the first set of packets, the first router receives the packet, decrements the TTL value and drops the packet because it then has TTL value zero. The router sends an ICMP Time Exceeded message back to the source. The next set of packets are given a TTL value of two, so the first router forwards the packets, but the second router drops them and replies with ICMP Time Exceeded.

Proceeding in this way, traceroute uses the returned ICMP Time Exceeded messages to build a list of routers that packets traverse, until the destination is reached and returns an ICMP Echo Reply message.

With the tracert command, we are asking tracert to show us the path from the local computer all the way to the network device with the hostname google.com.
> tracert google.com

6. ping:
The ping command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not. Tracking and isolating hardware and software problems. Determining the status of the network and various foreign hosts. The ping command is usually used as a simple way to verify that a computer can communicate over the network with another computer or network device. The ping command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to the destination computer and waiting for a response

ping172.16.6.2



Result

Thus, the different network commands available in Windows and Linux Operating Systems like tcpdump, netstat, ifconfig, nslookup and traceroute ping are executed successfully.

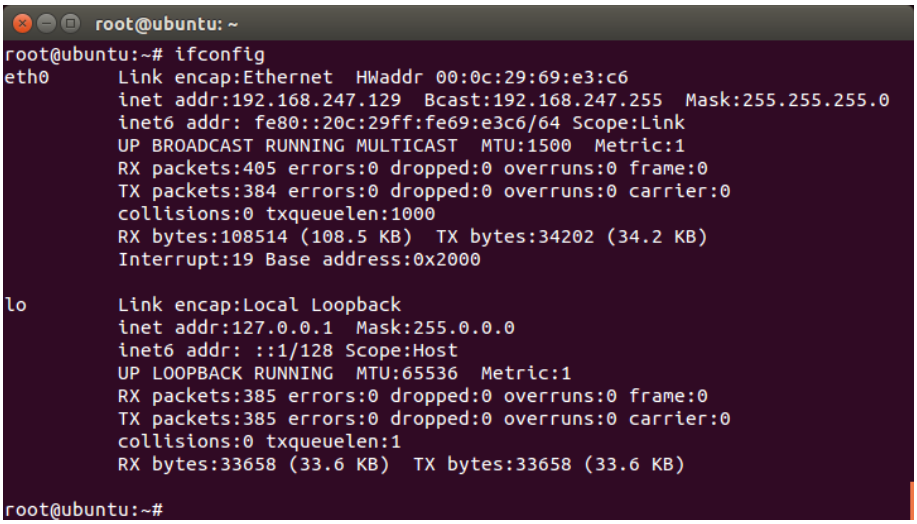
Ex. No: 02	Network configuration commands using Linux

Aim

To Learn to use Network configuration commands using Linux.

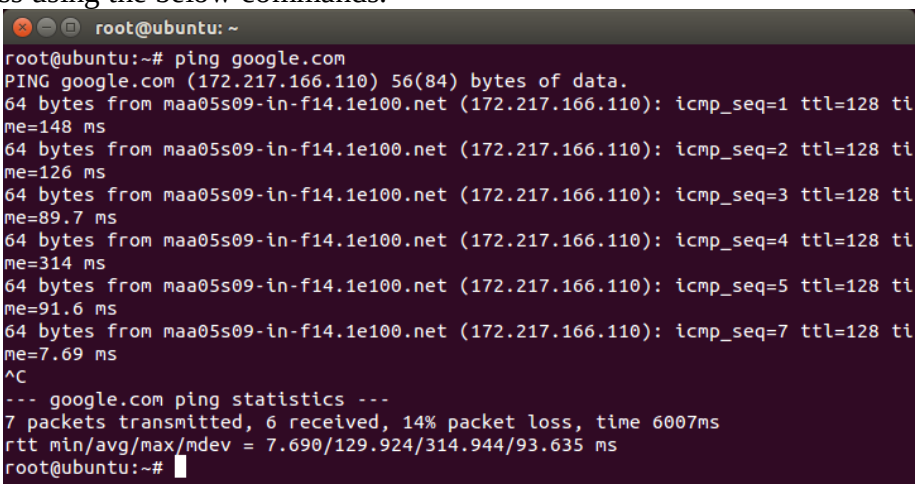
1. ifconfig Command

ifconfig (interface configurator) command is used to initialize an interface, assign IP Address to interface, and enable or disable interface on demand. With this command, you can view IP Address and Hardware / MAC address assign to interface and MTU (Maximum transmission unit) size.



2. Ping Command

Ping (Packet Internet Groper) command is the best way to test connectivity between two nodes. Whether it is Local Area Network (LAN) or Wide Area Network (WAN). Ping uses ICMP (Internet Control Message Protocol) to communicate to other devices. You can ping hostname or IP address using the below commands.



3. Traceroute Command

traceroute is a network troubleshooting utility that shows the number of hops taken to reach a destination also determines packets traveling path. Below we are tracing the route to the global

DNS server IP Address and able to reach destination also shows the path of that packet is traveling.

```
localhost:~# traceroute www.google.com
traceroute to www.google.com (142.251.46.164), 30 hops max, 38 byte packets
 1  10.5.0.1 (10.5.0.1)  399.000 ms  312.000 ms  270.000 ms
 2  172.17.0.1 (172.17.0.1)  396.000 ms  256.999 ms  277.000 ms
 3  *  *  *
 4  10.84.66.10 (10.84.66.10)  733.000 ms  10.84.66.14 (10.84.66.14)  269.000 ms
  10.84.66.12 (10.84.66.12)  270.000 ms
 5  *  *  *
 6  138.197.245.105 (138.197.245.105)  279.000 ms  303.000 ms  268.000 ms
 7  209.85.149.170 (209.85.149.170)  291.000 ms  138.68.33.9 (138.68.33.9)  295.
000 ms  209.85.149.170 (209.85.149.170)  268.000 ms
 8  *  *  *
 9  142.251.65.136 (142.251.65.136)  280.000 ms  142.251.65.128 (142.251.65.128)
  308.000 ms  142.251.224.30 (142.251.224.30)  819.000 ms
10  108.170.242.237 (108.170.242.237)  440.000 ms  142.251.67.63 (142.251.67.63)
  300.000 ms  276.000 ms
11  74.125.253.190 (74.125.253.190)  293.000 ms  nuq04s44-in-f4.1e100.net (142.2
51.46.164)  269.000 ms  272.000 ms
localhost:~#
```

4. Netstat Command

Netstat (Network Statistic) command displays connection info, routing table information, etc. To display routing table information use option as -r.

```
root@ubuntu: ~
root@ubuntu:~# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         192.168.247.2   0.0.0.0         UG        0 0          0 eth0
192.168.247.0   *               255.255.255.0   U         0 0          0 eth0
root@ubuntu:~#
```

5. Dig Command

Dig (domain information groper) query DNS related information like A Record, CNAME, MX Record, etc. This command is mainly used to troubleshoot DNS-related queries.

```
root@ubuntu: ~
root@ubuntu:~# dig www.google.com

; <<>> DiG 9.9.5-3ubuntu0.8-Ubuntu <<>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 63494
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; MBZ: 0005 , udp: 4000
;; QUESTION SECTION:
;www.google.com.                        IN      A

;; ANSWER SECTION:
www.google.com.         5       IN      A       142.250.77.164

;; Query time: 29 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Sun Nov 05 01:35:53 PST 2023
;; MSG SIZE  rcvd: 59

root@ubuntu:~#
```

6. Nslookup Command

nslookup command is also used to find out DNS-related queries. The following examples show A Record (IP Address) of tecmint.com

```
root@ubuntu: ~
root@ubuntu:~# nslookup google.com
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:   google.com
Address: 142.250.193.142

root@ubuntu:~#
```

7. Route Command

route command also shows and manipulates the ip routing table. To see the default routing table in Linux, type the following command.

```
root@ubuntu: ~
root@ubuntu:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.247.2   0.0.0.0         UG    0      0        0 eth0
192.168.247.0   *               255.255.255.0   U     1      0        0 eth0
root@ubuntu:~#
```

8. Host Command

host command to find a name to IP or IP to name in IPv4 or IPv6 and query DNS records.

```
root@ubuntu: ~
root@ubuntu:~# host www.google.com
www.google.com has address 142.250.77.164
www.google.com has IPv6 address 2404:6800:4007:829::2004
root@ubuntu:~#
```

9. Arp Command

ARP (Address Resolution Protocol) is useful to view/add the contents of the kernel ARP tables. To see the default table use the command as.

```
root@ubuntu: ~
root@ubuntu:~# arp -e
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.247.2            ether   00:50:56:e0:55:41   C                     eth0
192.168.247.254          ether   00:50:56:fe:1e:7d   C                     eth0
root@ubuntu:~#
```

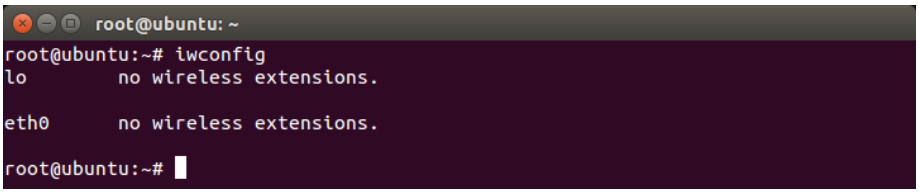
10. Ethtool Command

ethtool is a replacement for mii-tool. It is to view, setting speed and duplex of your Network Interface Card (NIC). You can set duplex permanently in /etc/sysconfig/network-scripts/ifcfg-eth0 with ETHTOOL_OPTS variable.

```
root@ubuntu: ~
root@ubuntu:~# ethtool eth0
Settings for eth0:
        Current message level: 0x00000007 (7)
                               drv probe link
        Link detected: yes
root@ubuntu:~#
```

11. Iwconfig Command

iwconfig command in Linux is used to configure a wireless network interface. You can see and set the basic Wi-Fi details like SSID channel and encryption. You can refer man page of iwconfig to know more.



12. Hostname Command

The hostname is to identify in a network. Execute the hostname command to see the hostname of your box. You can set hostname permanently in /etc/sysconfig/network. Need to reboot box once set a proper hostname.

Result

Thus, the various networks configuration commands are executed successfully.

Ex. No: 03	Error detection and correction mechanisms

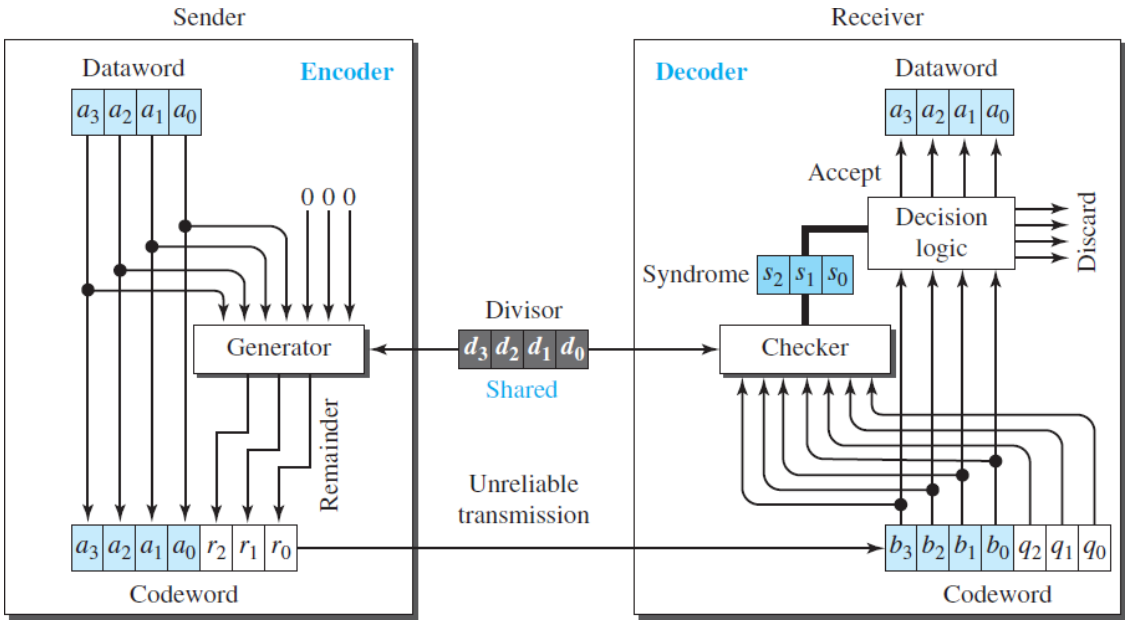
Aim

To Simulation of Error Correction Code using Java.

Theory

Cyclic Redundancy Check (CRC)

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



Example

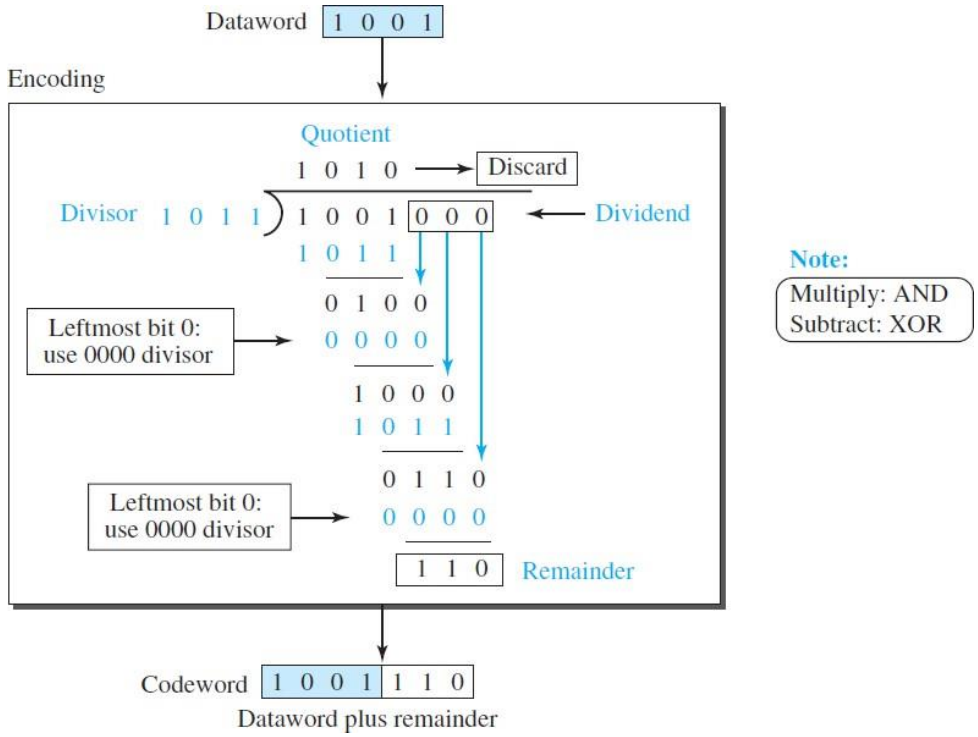


Figure: Division in the CRC encoder

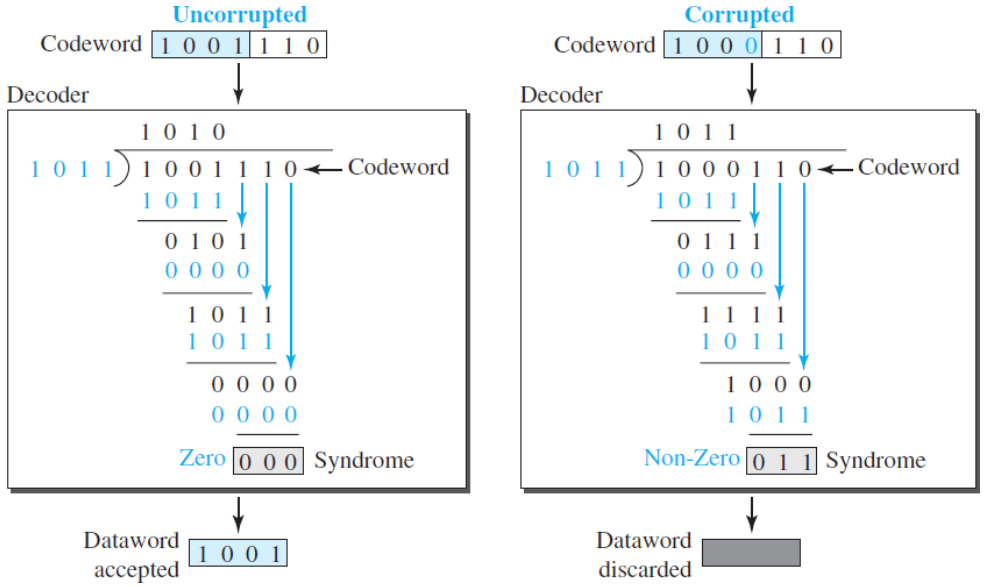


Figure: Division in the CRC decoder

Procedure

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

Program

```
import java.io.*;
class CRC
{
	public static void main(String args[]) throws IOException
	{
		BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
		System.out.println("Enter Generator:");
		String gen = br.readLine();
		System.out.println("Enter Data:");
		String data = br.readLine();
		String code = data;
		while(code.length() < (data.length() + gen.length() - 1)) code = code + "0";
		code = data + div(code,gen);
		System.out.println("The transmitted Code Word is: " + code);
		System.out.println("Please enter the received Code Word: ");
		String rec = br.readLine();
		if(Integer.parseInt(div(rec,gen)) == 0)
		System.out.println("The received code word contains no errors."); else
		System.out.println("The received code word contains errors.");
	}
	static String div(String num1,String num2)
	{
		int pointer = num2.length();
		String result = num1.substring(0, pointer); String remainder = "";
		for(int i = 0; i < num2.length(); i++)
		{
			if(result.charAt(i) == num2.charAt(i)) remainder += "0";
			else
			remainder += "1";
		}
		while(pointer < num1.length())
		{
			if(remainder.charAt(0) == '0')
			{
```

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

Output

```
Enter Generator:
1011
Enter Data:
1001
The transmitted Code Word is: 1001101
Please enter the received Code Word:
1001101
The received code word contains no errors.
```

Result

Thus, the error detection and error correction are implemented successfully.

Ex. No: 04 a	Flow control mechanisms Implementation of Sliding Window Protocol

Aim

To write a java program to perform sliding window protocol

Algorithm

1. Start the program.
2. Get the frame size from the user
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server, it will send ACK signal to client otherwise it will send NACK signal to client.
6. Stop the program

Program

Sender

```
import java.net.*;
import java.io.*;
import java.rmi.*;
public class slidsender
{
	public static void main(String a[])throws Exception
	{
		ServerSocket ser=new ServerSocket(10);
		Socket s=ser.accept();
		DataInputStream in=new DataInputStream(System.in);
		DataInputStream in1=new DataInputStream(s.getInputStream());
		String sbuff[]=new String[8];
		PrintStream p;
		int sptr=0,sws=8,nf,ano,i;
		String ch;
		do
		{
			p=new PrintStream(s.getOutputStream());
			System.out.println("Sender");
			System.out.print("Enter the no. of frames : ");
			nf=Integer.parseInt(in.readLine());
			p.println(nf);
			if(nf<=sws-1)
			{
				System.out.println("Enter "+nf+" Messages to be send\n");
				for(i=1;i<=nf;i++)
				{
```

```
                                        sbuff[sptr]=in.readLine();
                                        p.println(sbuff[sptr]);
                                        sptr=++sptr%8;
                                }
                                sws-=nf;
                                System.out.print("Acknowledgment received");
                                ano=Integer.parseInt(in1.readLine());
                                System.out.println(" for "+ano+" frames");
                                sws+=nf;
                        }
                        else
                        {
                                System.out.println("The no. of frames exceeds window size");
                                break;
                        }
                        System.out.print("\nDo you wants to send some more frames : ");
                        ch=in.readLine();
                        p.println(ch);
                }
                while(ch.equals("yes"));
                s.close();
        }
}
```

Receiver

```
import java.net.*;
import java.io.*;
class slidreceiver
{
        public static void main(String a[])throws Exception
        {
                Socket s=new Socket(InetAddress.getLocalHost(),10);
                DataInputStream in=new DataInputStream(s.getInputStream());
                PrintStream p=new PrintStream(s.getOutputStream());
                int i=0,rptr=-1,nf,rws=8;
                String rbuf[]=new String[8];
                String ch;
                System.out.println("Receiver");
                do
                {
                        nf=Integer.parseInt(in.readLine());
                        if(nf<=rws-1)
                        {
                                for(i=1;i<=nf;i++)
                                {
                                        rptr=++rptr%8; rbuf[rptr]=in.readLine();
                                        System.out.println("The received Frame " +rptr+" is :
"+rbuf[rptr]);
                                }
```

```
                                rws-=nf; System.out.println("\nAcknowledgment sent\n");
                                p.println(rptr+1); rws+=nf;
                        }
                        else break;
                        ch=in.readLine();
                }
                while(ch.equals("yes"));
        }
}
```

Output

```
Sender
Enter the no. of frames : 2
Enter 2 Messages to be send

Computer
Networks
Acknowledgment received for 2 frames

Do you wants to send some more frames :

Receiver
The received Frame 0 is : Computer
The received Frame 1 is : Networks

Acknowledgment sent
```

Result

Thus, the java program to perform sliding window protocol is implemented successfully

Ex. No: 04 b	Flow control mechanisms Implementation of Stop and Wait Protocol

Aim

To write a java program to perform sliding window protocol

Algorithm

1. Start the program.
2. Get the frame size from the user
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server, it will send ACK signal to client otherwise it will send NACK signal to client.
6. Stop the program

Program

Sender

```
import java.io.*;
import java.net.*;
public class Sender
{
	Socket sender;
	ObjectOutputStream out;
	ObjectInputStream in;
	String packet,ack,str, msg;
	int n,i=0,sequence=0;
	Sender()
	{}
	public void run()
	{
		try
		{
			BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
			System.out.println("Sender");
			System.out.println("Waiting for Connection. .. ");
			sender = new Socket("localhost",2004);
			sequence=0;
			out=new ObjectOutputStream(sender.getOutputStream());
			out.flush();
			in=new ObjectInputStream(sender.getInputStream());
			str=(String)in.readObject();
			System.out.println("reciver > "+str);
			System.out.println("Enter the data to send. . ");
```

```
                    packet=br.readLine();
                    n=packet.length();
                    do
                    {
                        try
                        {
                            if(i<n)
                            {
                                msg=String.valueOf(sequence);
                                msg=msg.concat(packet.substring(i,i+1));
                            }
                            else if(i==n)
                            {
                                msg="end";
                                out.writeObject(msg);
                                break;
                            }
                            out.writeObject(msg);
                            sequence=(sequence==0)?1:0;
                            out.flush();
                            System.out.println("data sent>"+msg);
                            ack=(String)in.readObject();
                            System.out.println("waiting for ack. ....\n\n");
                            if(ack.equals(String.valueOf(sequence)))
                            {
                                i++;
                                System.out.println("receiver > "+" packet
recieved\n\n");
                            }
                            else
                            {
                                System.out.println("Time out resending data ...
\n\n");
                                sequence=(sequence==0)?1:0;
                            }
                        }
                        catch(Exception e)
                        {}
                    }
                    while(i<n+1);
                    System.out.println("All data sent. exiting.");
                }
                catch(Exception e)
                {}
                finally
                {
                    try
                    {
                        in.close();
                        out.close();
```

```
                        sender.close();
                    }
                    catch(Exception e){}
                }
        }
        public static void main(String args[])
        {
                Sender s=new Sender();
                s.run();
        }
}
```

Receiver

```
import java.io.*;
import java.net.*;
public class Reciever
{
        ServerSocket reciever;
        Socket connection=null;
        ObjectOutputStream out;
        ObjectInputStream in;
        String packet,ack,data="";
        int i=0,sequence=0;
        Reciever(){}
        public void run()
        {
                try
                {
                        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
                        reciever = new ServerSocket(2004,10);
                        System.out.println("Receiver");
                        System.out.println("waiting for connection...");
                        connection=reciever.accept();
                        sequence=0;
                        System.out.println("Connection established :");
                        out=new ObjectOutputStream(connection.getOutputStream());
                        out.flush();
                        in=new ObjectInputStream(connection.getInputStream());
                        out.writeObject("connected .");
                        do
                        {
                                try
                                {
                                        packet=(String)in.readObject();
                                        if(Integer.valueOf(packet.substring(0,1))==sequence)
                                        {
                                                data+=packet.substring(1);
                                                sequence=(sequence==0)?1:0;
```

```
                                        System.out.println("\n\nreceiver >"+packet);
                                }
                                else
                                {
                                        System.out.println("\n\nreceiver >"+packet +"
duplicate data");
                                }
                                if(i<3)
                                {
                                        out.writeObject(String.valueOf(sequence));
                                        i++;
                                }
                                else
                                {

        out.writeObject(String.valueOf((sequence+1)%2));
                                        i=0;
                                }
                        }
                        catch(Exception e){}
                }
                while(!packet.equals("end"));
                System.out.println("Data received="+data);
                out.writeObject("connection ended .");
        }
        catch(Exception e){}
        finally
        {
                try
                {
                        in.close();
                        out.close();
                        reciever.close();
                }
                catch(Exception e){}
        }
    }
    public static void main(String args[])
    {
        Reciever s=new Reciever();
        while(true)
        {
                s.run();
        }
    }
}
```

Output

Sender

```
Waiting for Connection. ..
receiver > connected .
Enter the data to send. .
RMKCET
data sent>0R
waiting for ack. ...
receiver >  packet received
data sent>1M
waiting for ack. ...
receiver >  packet received
data sent>0K
waiting for ack. ...
receiver >  packet received
data sent>1C
waiting for ack. ...
Time out resending data ...
data sent>1C
waiting for ack. ...
receiver >  packet received
data sent>0E
waiting for ack. ...
receiver >  packet received
data sent>1T
waiting for ack. ...
receiver >  packet received
All data sent. exiting.
Press any key to continue . . .
```

Receiver

```
waiting for connection...
Connection established :
receiver  >0R
receiver >1M
receiver  >0K
receiver >1C
receiver >1C duplicate data
receiver >0E
receiver >1T
Data received=RMKCET
Receiver
waiting for connection...
```

Result

Thus, the flow control mechanism is implemented successfully.

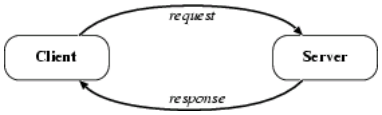
Ex. No: 05	Multi-client chatting in TCP and UDP using Socket programming

Aim

To write a java program for application using TCP Sockets Links

Procedure

- In the TCP Echo client, a socket is created. Using the socket a connection is made to the server using the connect() function. After a connection is established, we send messages input from the user and display the data received from the server using send() and read() functions.
- In the TCP Echo server, we create a socket and bind to an advertised port number. After binding the process listens for incoming connections. Then an infinite loop is started to process the client requests for connections. After a connection is requested, it accepts the connection from the client machine and forks a new process.
- The new process receives data from the client using recv() function and echoes the same data using the send() function. Please note that this server is capable of handling multiple clients as it forks a new process for every client trying to connect to the server. TCP socket routines enable reliable IP communication using the transmission control protocol (TCP).
- The implementation of the Transmission Control Protocol (TCP) in the Network Component. TCP runs on top of the Internet Protocol (IP). TCP is a connection-oriented and reliable, full duplex protocol supporting a pair of byte streams, one for each direction.
- A TCP connection must be established before exchanging data. TCP retransmits data that do not reach the destination due to errors or data corruption. Data is delivered in the sequence of its transmission.



a. <u>Echo client and echo server</u>

Algorithm

Client

1. Start
2. Create the TCP socket
3. Establish connection with the server
4. Get the message to be echoed from the user
5. Send the message to the server
6. Receive the message echoed by the server
7. Display the message received from the server
8. Terminate the connection
9. Stop

Server

1. Start
2. Create TCP socket, make it a listening socket
3. Accept the connection request sent by the client for connection establishment
4. Receive the message sent by the client
5. Display the received message
6. Send the received message to the client from which it receives
7. Close the connection when client initiates termination and server become a listening server, waiting for clients.
8. Stop.

Program

Server

```
import java.io.*;
import java.net.*;
public class EchoServer
{
	public static void main(String[] args)
	{
		ServerSocket serverSocket = null;
		try
		{
			serverSocket = new ServerSocket(9000);
			System.out.println("Server listening on port 9000...");
		}
		catch (IOException e)
		{
			System.out.println("Error: " + e.getMessage());
			return;
		}
		while (true)
		{
			try (Socket clientSocket = serverSocket.accept();
			BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
			PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
true))
			{
				System.out.println("Client connected: " +
clientSocket.getInetAddress().getHostAddress());
				String inputLine;
				while ((inputLine = in.readLine()) != null)
				{
					System.out.println("Received from client: " +
inputLine);
					out.println("Server: " + inputLine);
				}
```

```
                                System.out.println("Client disconnected: " +
clientSocket.getInetAddress().getHostAddress());
                        }
                        catch (IOException e)
                        {
                                System.out.println("Error: " + e.getMessage());
                        }
                }
        }
}
```

Client

```
import java.io.*;
import java.net.*;
public class EchoClient
{
        public static void main(String[] args)
        {
                try (Socket socket = new Socket("localhost", 9000);
                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
                BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in)))
                {
                        System.out.println("Connected to server.");
                        String userInput;
                        while (true)
                        {
                                System.out.print("Client: ");
                                userInput = reader.readLine();
                                out.println(userInput);
                                String serverResponse = in.readLine();
                                System.out.println("Server: " + serverResponse);
                                if (userInput.equalsIgnoreCase("quit"))
                                {
                                        System.out.println("Disconnected from server.");
                                        break;
                                }
                        }
                }
                catch (IOException e)
                {
                        System.out.println("Error: " + e.getMessage());
                }
        }
}
```

Output

Client

Connected to server.
Client: hai server
Server: Server: hai server
Client: hello server
Server: Server: hello server
Client: quit
Server: Server: quit
Disconnected from server.
Press any key to continue . . .

Server

Server listening on port 9000...
Client connected: 127.0.0.1
Received from client: hai server
Received from client: hello server
Received from client: quit
Client disconnected: 127.0.0.1

b. Chat

Algorithm

Client

1. Start
2. Create the UDP datagram socket
3. Get the request message to be sent from the user
4. Send the request message to the server
5. If the request message is ―END‖ go to step 10
6. Wait for the reply message from the server
7. Receive the reply message sent by the server
8. Display the reply message received from the server
9. Repeat the steps from 3 to 8
10. Stop

Server

1. Start
2. Create UDP datagram socket, make it a listening socket
3. Receive the request message sent by the client
4. If the received message is ”END” go to step 10
5. Retrieve the client ‘s IP address from the request message received
6. Display the received message
7. Get the reply message from the user
8. Send the reply message to the client

9. Repeat the steps from 3 to 8.
10. Stop.

Program

Server

```
import java.io.*;
import java.net.*;
class UDPserver
{
	public static DatagramSocket ds;
	public static byte buffer[]=new byte[1024];
	public static int clientport=789,serverport=790;
	public static void main(String args[])throws Exception
	{
		ds=new DatagramSocket(clientport);
		System.out.println("press ctrl+c to quit the program");
		BufferedReader dis=new BufferedReader(new
InputStreamReader(System.in));
		InetAddress ia=InetAddress.getLocalHost();
		while(true)
		{
			DatagramPacket p=new DatagramPacket(buffer,buffer.length);
			ds.receive(p);
			String psx=new String(p.getData(),0,p.getLength());
			System.out.println("Client:" + psx);
			System.out.println("Server:");
			String str=dis.readLine();
			if(str.equals("end"))
			break;
			buffer=str.getBytes();
			ds.send(new DatagramPacket(buffer,str.length(),ia,serverport));
		}
	}
}
```

Client

```
import java .io.*;
import java.net.*;
class UDPclient
{
	public static DatagramSocket ds;
	public static int clientport=789,serverport=790;
	public static void main(String args[])throws Exception
	{
		byte buffer[]=new byte[1024];
		ds=new DatagramSocket(serverport);
```

```
                BufferedReader dis=new BufferedReader(new
InputStreamReader(System.in));
                System.out.println("server waiting");
                InetAddress ia=InetAddress.getLocalHost();
                while(true)
                {
                        System.out.println("Client:");
                        String str=dis.readLine();
                        if(str.equals("end"))
                        break;
                        buffer=str.getBytes();
                        ds.send(new DatagramPacket(buffer,str.length(),ia,clientport));
                        DatagramPacket p=new DatagramPacket(buffer,buffer.length);
                        ds.receive(p);
                        String psx=new String(p.getData(),0,p.getLength());
                        System.out.println("Server:" + psx);
                }
        }
}
```

Output

Client

```
server waiting
Client:
hai
Server:hel
Client:
end
Press any key to continue . . .
```

Server

```
press ctrl+c to quit the program
Client:hai
Server:
hello
```

Result

Thus, the multi-client chatting in TCP and UDP using Socket programming is executed successfully.

Ex. No: 06	Implementation of HTTP, Web Caching, FTP using socket programming

Aim

To write a java program for socket for HTTP for web page upload and download.

Procedure

- HTTP means Hyper Text Transfer Protocol. HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
- For example, when you enter a URL in your browser, this sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page.
- The other main standard that controls how the World Wide Web works is HTML, which covers how Web pages are formatted and displayed. HTTP functions as a request–response protocol in the client–server computing model.
- A web browser, for example, may be the client and an application running on a computer hosting a website may be the server.
- The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content, or performs other functions on behalf of the client, returns a response message to the client.
- The response contains completion status information about the request and may also contain requested content in its message body.

Algorithm

Client

1. Start.
2. Create socket and establish the connection with the server.
3. Read the image to be uploaded from the disk
4. Send the image read to the server
5. Terminate the connection
6. Stop.

Server

1. Start
2. Create socket, bind IP address, and port number with the created socket and make server a listening server.
3. Accept the connection request from the client
4. Receive the image sent by the client.
5. Display the image.
6. Close the connection.
7. Stop.

Program

Client

```
import javax.swing.*;
import java.net.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
public class Client
{
	public static void main(String args[]) throws Exception
	{
		Socket soc;
		BufferedImage img = null;
		soc=new Socket("localhost",4000);
		System.out.println("Client is running");
		try
		{
			System.out.println("Reading image from disk. ");
			img = ImageIO.read(new File("1.png"));
			ByteArrayOutputStream baos = new ByteArrayOutputStream();
			ImageIO.write(img, "jpg", baos);
			baos.flush();
			byte[] bytes = baos.toByteArray();
			baos.close();
			System.out.println("Sending image to server.");
			OutputStream out = soc.getOutputStream();
			DataOutputStream dos = new DataOutputStream(out);
			dos.writeInt(bytes.length);
			dos.write(bytes, 0, bytes.length);
			System.out.println("Image sent to server. ");
			dos.close();
			out.close();
		}
		catch (Exception e)
		{
			System.out.println("Exception: " + e.getMessage());
			soc.close();
		}
		soc.close();
	}
}
```

Server

```
import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;
class Server
{
	public static void main(String args[]) throws Exception
	{
		ServerSocket server=null;
		Socket socket;
		server=new ServerSocket(4000);
		System.out.println("Server Waiting for image");
		socket=server.accept();
		System.out.println("Client connected.");
		InputStream in = socket.getInputStream();
		DataInputStream dis = new DataInputStream(in);
		int len = dis.readInt();
		System.out.println("Image Size: " + len/1024 + "KB");
		byte[] data = new byte[len];
		dis.readFully(data);
		dis.close();
		in.close();
		InputStream ian = new ByteArrayInputStream(data);
		BufferedImage bImage = ImageIO.read(ian);
		JFrame f = new JFrame("Server");
		ImageIcon icon = new ImageIcon(bImage);
		JLabel l = new JLabel();
		l.setIcon(icon);
		f.add(l);
		f.pack();
		f.setVisible(true);
	}
}
```

Output

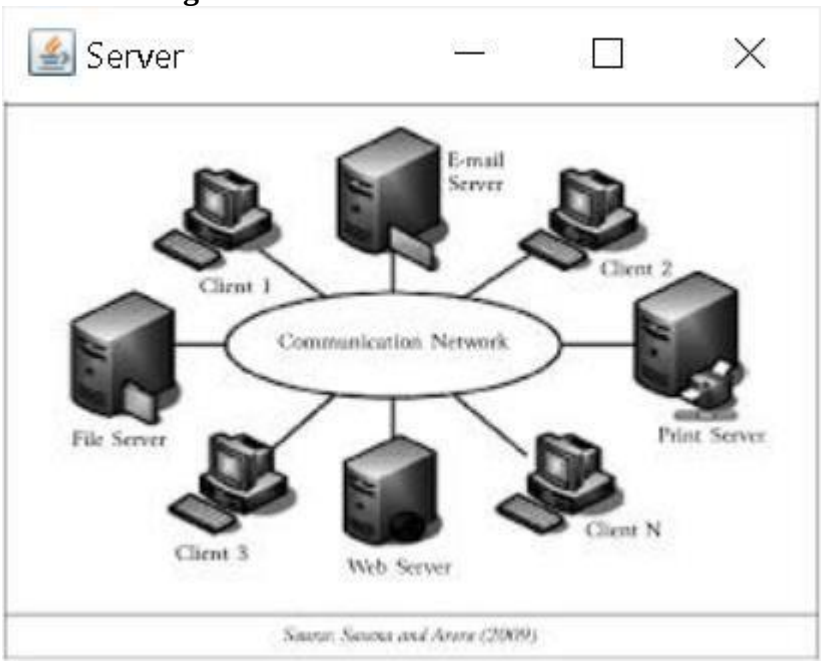
Client

Client is running
Reading image from disk.
Sending image to server.
Image sent to server.
Press any key to continue . . .

Server

Server Waiting for image
Client connected.
Image Size: 11KB

Received image



Result

Thus, socket for HTTP for web page upload and download is executed successfully.

Ex. No: 07	**Develop a DNS client server to resolve the given host name or IP address**

Aim

To write a java program for DNS application

Procedure

- The Domain Name System (DNS) is a hierarchical decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities.
- The domain name space refers a hierarchy in the internet naming structure. This hierarchy has multiple levels (from 0 to 127), with a root at the top. The following diagram shows the domain name space hierarchy.
- Name server contains the DNS database. This database comprises of various names and their corresponding IP addresses. Since it is not possible for a single server to maintain entire DNS database, therefore, the information is distributed among many DNS servers.
- Types of Name Servers
- Root Server is the top-level server which consists of the entire DNS tree. It does not contain the information about domains but delegates the authority to the other server
- Primary Server stores a file about its zone. It has authority to create, maintain, and update the zone file.
- Secondary Server transfers complete information about a zone from another server which may be primary or secondary server. The secondary server does not have authority to create or update a zone file.
- DNS is a TCP/IP protocol used on different platforms. The domain name space is divided into three different sections: generic domains, country domains, and inverse domain.
- The main function of DNS is to translate domain names into IP Addresses, which computers can understand. It also provides a list of mail servers which accept Emails for each domain name. Each domain name in DNS will nominate a set of name servers to be authoritative for its DNS records.

Algorithm

Server

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

Client

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

Program

Server

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

Client

```
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();
        }
}
```

Output

Server

```
Press Ctrl + C to Quit
Request for host yahoo.com
```

Client

```
Enter the hostname: yahoo.com
IP Address: 68.180.206.184
Press any key to continue . . .
```

Result

Thus, the java application program using UDP Sockets to implement DNS was developed and executed successfully.

Ex. No: 08	Simulation of unicast routing protocols

Aim

To write a NS2 program for implementing unicast routing protocol.

Procedure

- When a device has multiple paths to reach a destination, it always selects one path by preferring it over others. This selection process is termed as Routing. Routing is done by special network devices called routers or it can be done by means of software processes.
- The software-based routers have limited functionality and limited scope. A router is always configured with some default route. A default route tells the router where to forward a packet if there is no route found for specific destination.
- In case there are multiple paths existing to reach the same destination, router can make decision based on the following information. Routes can be statically configured or dynamically learnt. One route can be configured to be preferred over others. Most of the traffic on the internet and intranets known as unicast data or unicast traffic is sent with specified destination. Routing unicast data over the internet is called unicast routing.
- It is the simplest form of routing because the destination is already known. Hence the router just must look up the routing table and forward the packet to next hop.
- Multicasting in computer network is a group communication, where a sender(s) send data to multiple receivers simultaneously. It supports one – to – many and many – to – many data transmission across LANs or WANs. Through the process of multicasting, the communication and processing overhead of sending the same data packet or data frame in minimized.
- Multicast IP Routing protocols are used to distribute data (for example, audio/video streaming broadcasts) to multiple recipients. Using multicast, a source can send a single copy of data to a single multicast address, which is then distributed to an entire group of recipients.
- The key difference between broadcast and multicast is that in the broadcast the packet is delivered to all the host connected to the network whereas, in multicast packet is delivered to intended recipients only.
- Multicast Message. Multicasting identifies logical groups of computers. A single message can then be sent to the group. Multicast Message. Multicasting uses the Internet Group Management Protocol (IGMP) to identify groups and group members.

Algorithm

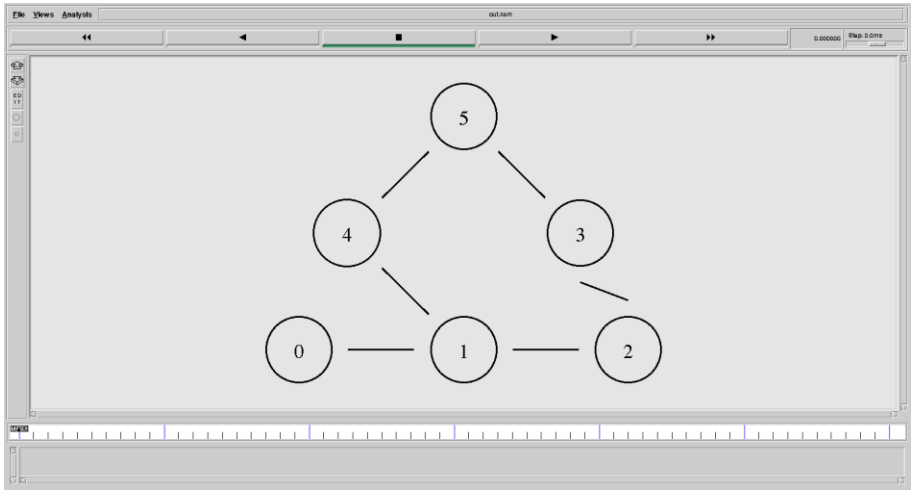
1. Start the program.
2. Declare the global variables ns for creating a new simulator.
3. Set the colour for packets.
4. Open the network animator file in the name of file2 in the write mode.
5. Open the trace file in the name of file 1 in the write mode.
6. Set the unicast routing protocol to transfer the packets in network.
7. Create the required no of nodes.
8. Create the duplex-link between the nodes including the delay time, bandwidth and dropping queue mechanism.
9. Give the position for the links between the nodes.
10. Set a TCP reno connection for source node.
11. Set the destination node using TCP sink.
12. Setup an ftp connection over the TCP connection.
13. Down the connection between any nodes at a particular time.
14. Reconnect the downed connection at a particular time.
15. Define the finish procedure.
16. In the definition of the finish procedure declare the global variables ns, file1, and file2.
17. Close the trace file and name file and execute the network animation file.
18. At the time call the finish procedure.
19. Stop the program.

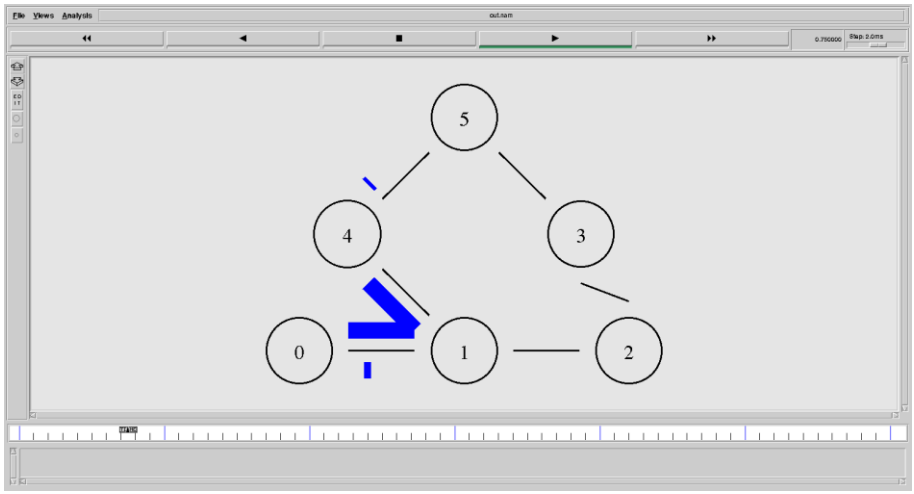
Program

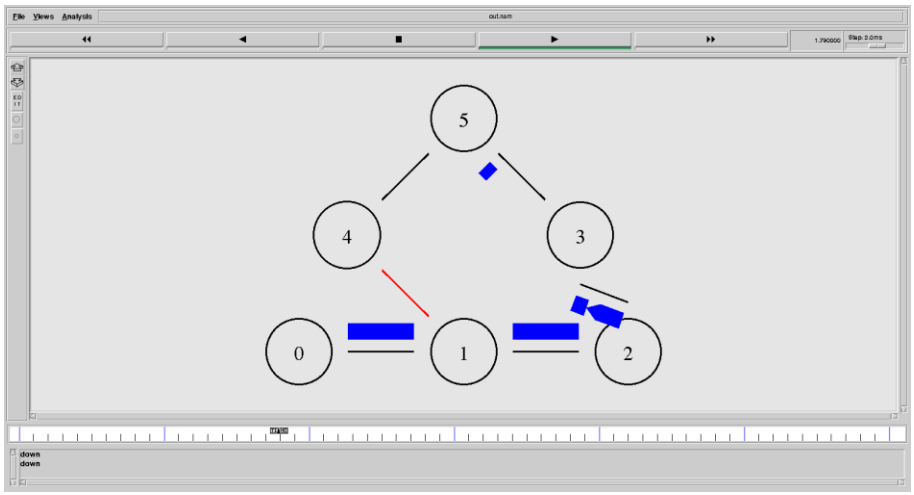
```
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red
#Open the Trace file
set file1 [open out.tr w]
$ns trace-all $file1

#Open the NAM trace file
set file2 [open out.nam w]
$ns namtrace-all $file2
#Define a 'finish' procedure
proc finish {} \
{
global ns file1 file2
$ns flush-trace
close $file1
close $file2
exec nam out.nam &
exit 3
}

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

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

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

#Give node position (for NAM)
$ns duplex-link-op $n0 $n1 orient right
$ns duplex-link-op $n1 $n2 orient right
$ns duplex-link-op $n2 $n3 orient up
$ns duplex-link-op $n1 $n4 orient up-left
$ns duplex-link-op $n3 $n5 orient left-up
$ns duplex-link-op $n4 $n5 orient right-up

#Setup a TCP connection
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

#Setup an FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
$ns rtmodel-at 1.0 down $n1 $n4
$ns rtmodel-at 4.5 up $n1 $n4
$ns at 0.1 "$ftp start"
$ns at 6.0 "finish"
$ns run
```

Output







Result

Thus, unicast routing protocols was developed and executed successfully.

Ex. No: 09	Observing Packets across the network and Performance Analysis of various Routing protocols

Aim

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

Pre Lab-Discussion

Link State Routing

Routing is the process of selecting best paths in a network. In the past, the term routing was also used to mean forwarding network traffic among networks. However, this latter function is much better described as simply forwarding. Routing is performed for many kinds of networks, including the telephone network (circuit switching), electronic data networks (such as the Internet), and transportation networks. This article is concerned primarily with routing in electronic data networks using packet switching technology.

In packet switching networks, routing directs packet forwarding (the transit of logically addressed network packets from their source toward their ultimate destination) through intermediate nodes. Intermediate nodes are typically network hardware devices such as routers, bridges, gateways, firewalls, or switches. General-purpose computers can also forward packets and perform routing, though they are not specialized hardware and may suffer from limited performance. The routing process usually directs forwarding based on routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the router's memory, is very important for efficient routing. Most routing algorithms use only one network path at a time. Multipath routing techniques enable the use of multiple alternative paths.

In case of overlapping/equal routes, the following elements are considered to decide which routes get installed into the routing table (sorted by priority):

1. Prefix-Length: where longer subnet masks are preferred (independent of whether it is within a routing protocol or over different routing protocol)
2. Metric: where a lower metric/cost is preferred (only valid within one and the same routing protocol)
3. Administrative distance: where a lower distance is preferred (only valid between different routing protocols)

Routing, in a narrower sense of the term, is often contrasted with bridging in its assumption that network addresses are structured and that similar addresses imply proximity within the network. Structured addresses allow a single routing table entry to represent the route to a group of devices. In large networks, structured addressing (routing, in the narrow sense) outperforms unstructured addressing (bridging). Routing has become the dominant form of addressing on the Internet. Bridging is still widely used within localized environments.

Flooding

Flooding is a simple routing algorithm in which every incoming packet is sent through every outgoing link except the one it arrived on. Flooding is used in bridging and in systems such as Usenet and peer-to-peer file sharing and as part of some routing protocols, including OSPF,

DVMRP, and those used in ad-hoc wireless networks. There are generally two types of flooding available, Uncontrolled Flooding and Controlled Flooding. Uncontrolled Flooding is the fatal law of flooding. All nodes have neighbours and route packets indefinitely. More than two neighbours create a broadcast storm.

Controlled Flooding has its own two algorithms to make it reliable, SNCF (Sequence Number Controlled Flooding) and RPF (Reverse Path Flooding). In SNCF, the node attaches its own address and sequence number to the packet, since every node has a memory of addresses and sequence numbers. If it receives a packet in memory, it drops it immediately while in RPF, the node will only send the packet forward. If it is received from the next node, it sends it back to the sender.

Distance Vector Routing

In computer communication theory relating to packet-switched networks, a distance- vector routing protocol is one of the two major classes of routing protocols, the other major class being the link-state protocol. Distance-vector routing protocols use the Bellman–Ford algorithm, Ford–Fulkerson algorithm, or DUAL FSM (in the case of Cisco Systems protocols) to calculate paths.

A distance-vector routing protocol requires that a router informs its neighbours of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.

The term distance vector refers to the fact that the protocol manipulates vectors (arrays) of distances to other nodes in the network. The vector distance algorithm was the original ARPANET routing algorithm and was also used in the internet under the name of RIP (Routing Information Protocol). Examples of distance-vector routing protocols include RIPv1 and RIPv2 and IGRP.

Method

Routers using distance-vector protocol do not have knowledge of the entire path to a destination. Instead, they use two methods:

1. Direction in which router or exit interface a packet should be forwarded.
2. Distance from its destination.

Distance-vector protocols are based on calculating the direction and distance to any link in a network. "Direction" usually means the next hop address and the exit interface. "Distance" is a measure of the cost to reach a certain node. The least cost route between any two nodes is the route with minimum distance. Each node maintains a vector (table) of minimum distance to every node. The cost of reaching a destination is calculated using various route metrics. RIP uses the hop count of the destination whereas IGRP considers other information such as node delay and available bandwidth.

Updates are performed periodically in a distance-vector protocol where all or part of a router's routing table is sent to all its neighbours that are configured to use the same distance- vector routing protocol. RIP supports cross-platform distance vector routing whereas IGRP is a Cisco Systems proprietary distance vector routing protocol. Once a router has this information it can amend its own routing table to reflect the changes and then inform its neighbours of the changes. This process has been described as routing by rumor because routers are relying on

the information they receive from other routers and cannot determine if the information is valid and true. There are several features which can be used to help with instability and inaccurate routing information.

EGP and BGP are not pure distance-vector routing protocols because a distance-vector protocol calculates routes based only on link costs whereas in BGP, for example, the local route preference value takes priority over the link cost.

Count-to-infinity problem

The Bellman–Ford algorithm does not prevent routing loops from happening and suffers from the count-to-infinity problem. The core of the count-to-infinity problem is that if A tells B that it has a path somewhere, there is no way for B to know if the path has B as a part of it. To see the problem clearly, imagine a subnet connected like A–B–C–D–E–F, and let the metric between the routers be "number of jumps". Now suppose that A is taken offline. In the vector-update-process B notices that the route to A, which was distance 1, is down – B does not receive the vector update from A. The problem is, B also gets an update from C, and C is still not aware of the fact that A is down – so it tells B that A is only two jumps from C (C to B to A), which is false. This slowly propagates through the network until it reaches infinity (in which case the algorithm corrects itself, due to the relaxation property of Bellman–Ford).

Algorithm

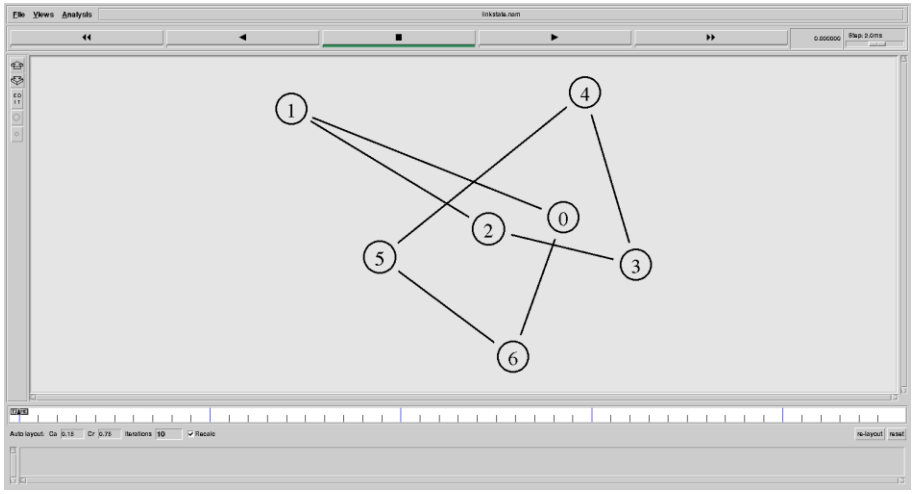
1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes.
7. Create the applications and attach them to the udp agent.
8. Connect udp and null.
9. At 1 sec the link between node 1 and 2 is broken.
10. At 2 sec the link is up again.
11. Run the simulation.

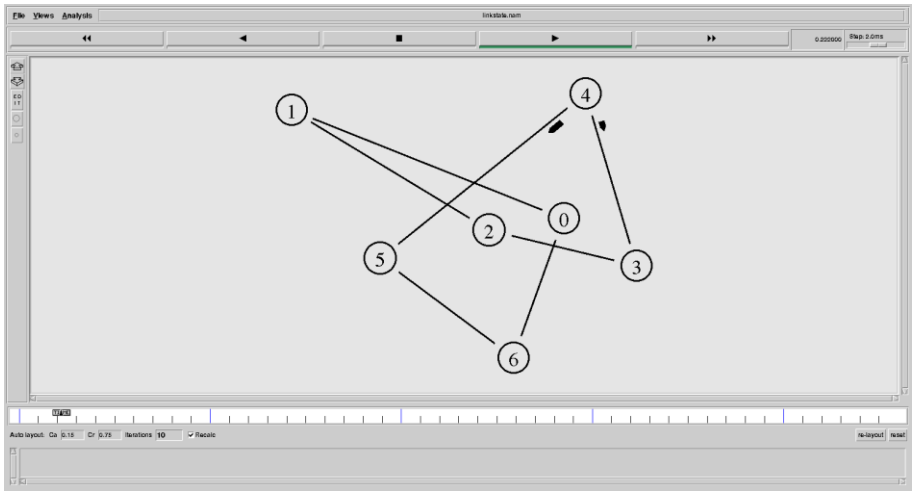
Link State Routing Protocol

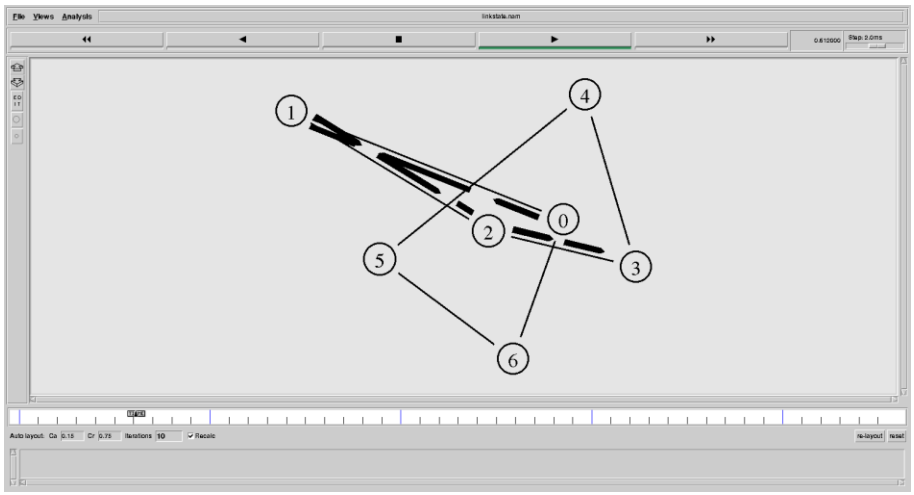
Program

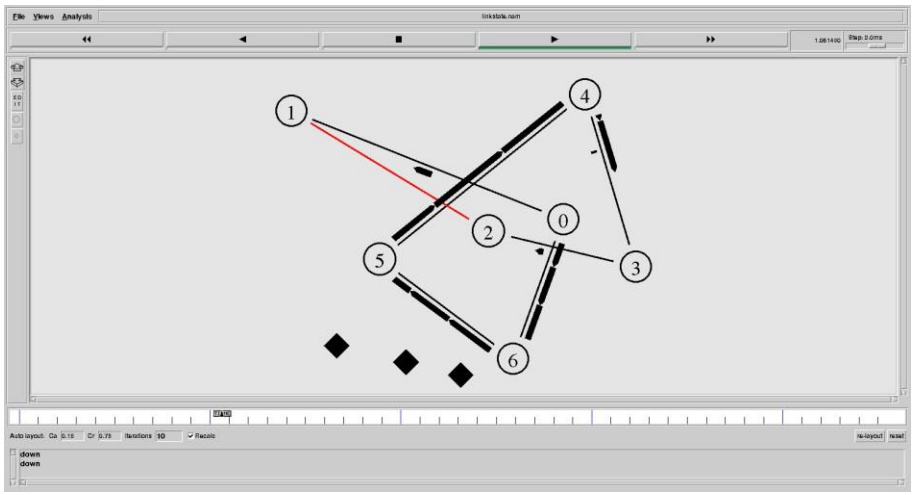
```
set ns [new Simulator]
$ns rtproto LS
set nf [open linkstate.nam w]
$ns namtrace-all $nf
set f0 [open linkstate.tr w]
$ns trace-all $f0
proc finish {} \
{
global ns f0 nf
$ns flush-trace
close $f0
close $nf
exec nam linkstate.nam &
exit 0
}
for {set i 0} {$i <7} {incr i} {
set n($i) [$ns node]
}
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









linkstate.tr (~/ns) - gedit

Open Save Undo

linkstate.tr

```
+ 0.00017 0 1 rtProtoLS 100 ------- 0 0.2 1.1 -1 0
- 0.00017 0 1 rtProtoLS 100 ------- 0 0.2 1.1 -1 0
+ 0.00017 0 6 rtProtoLS 100 ------- 0 0.2 6.1 -1 1
- 0.00017 0 6 rtProtoLS 100 ------- 0 0.2 6.1 -1 1
+ 0.007102 2 1 rtProtoLS 100 ------- 0 2.1 1.1 -1 2
- 0.007102 2 1 rtProtoLS 100 ------- 0 2.1 1.1 -1 2
+ 0.007102 2 3 rtProtoLS 100 ------- 0 2.1 3.2 -1 3
- 0.007102 2 3 rtProtoLS 100 ------- 0 2.1 3.2 -1 3
r 0.01097 0 1 rtProtoLS 100 ------- 0 0.2 1.1 -1 0
+ 0.01097 1 0 rtProtoLS 20 ------- 0 1.1 0.2 -1 4
- 0.01097 1 0 rtProtoLS 20 ------- 0 1.1 0.2 -1 4
+ 0.01097 1 2 rtProtoLS 100 ------- 0 1.1 2.1 -1 5
- 0.01097 1 2 rtProtoLS 100 ------- 0 1.1 2.1 -1 5
r 0.01097 0 6 rtProtoLS 100 ------- 0 0.2 6.1 -1 1
+ 0.01097 6 0 rtProtoLS 20 ------- 0 6.1 0.2 -1 6
- 0.01097 6 0 rtProtoLS 20 ------- 0 6.1 0.2 -1 6
+ 0.01097 6 5 rtProtoLS 100 ------- 0 6.1 5.1 -1 7
- 0.01097 6 5 rtProtoLS 100 ------- 0 6.1 5.1 -1 7
r 0.017902 2 1 rtProtoLS 100 ------- 0 2.1 1.1 -1 2
+ 0.017902 1 2 rtProtoLS 20 ------- 0 1.1 2.1 -1 8
- 0.017902 1 2 rtProtoLS 20 ------- 0 1.1 2.1 -1 8
+ 0.017902 1 0 rtProtoLS 100 ------- 0 1.1 0.2 -1 9
- 0.017902 1 0 rtProtoLS 100 ------- 0 1.1 0.2 -1 9
```

Plain Text Tab Width: 8 Ln 12, Col 51 INS

Distance Vector Routing Algorithm

Algorithm

1. Create a simulator object
2. Set routing protocol to Distance Vector routing
3. Trace packets on all links onto NAM trace and text trace file
4. Define finish procedure to close files, flush tracing and run NAM
5. Create eight nodes
6. Specify the link characteristics between nodes
7. Describe their layout topology as a octagon
8. Add UDP agent for node n1
9. Create CBR traffic on top of UDP and set traffic parameters.
10. Add a sink agent to node n4
11. Connect source and the sink
12. Schedule events as follows:
    a. Start traffic flow at 0.5
    b. Down the link n3-n4 at 1.0
    c. Up the link n3-n4 at 2.0
    d. Stop traffic at 3.0
    e. Call finish procedure at 5.0
13. Start the scheduler
14. Observe the traffic route when link is up and down
15. View the simulated events and trace file analyze it
16. Stop

Program

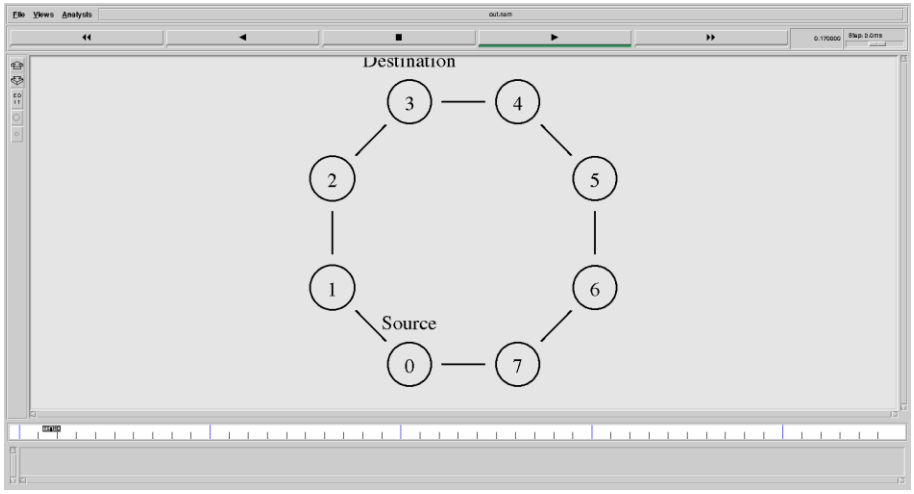
```
#Distance vector routing protocol – distvect.tcl
#Create a simulator object
set ns [new Simulator]

#Use distance vector routing
$ns rtproto DV

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

#Define 'finish' procedure
proc finish {} \
{
global ns nf
$ns flush-trace
#Close the trace file
close $nf
```

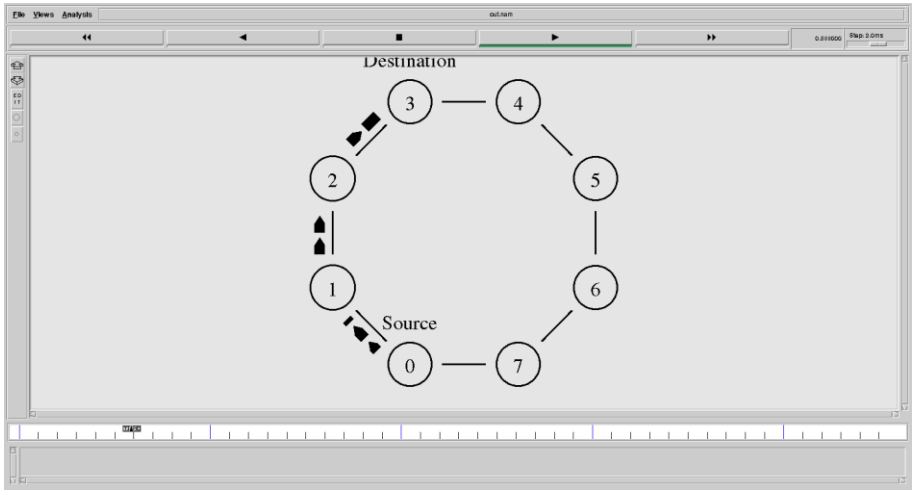
```
#Execute nam on the trace file
exec nam -a out.nam &
exit 0
}
# Create 8 nodes
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]

# Specify link characterestics
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns duplex-link $n5 $n6 1Mb 10ms DropTail
$ns duplex-link $n6 $n7 1Mb 10ms DropTail
$ns duplex-link $n7 $n8 1Mb 10ms DropTail
$ns duplex-link $n8 $n1 1Mb 10ms DropTail

# Specify layout as an octagon
$ns duplex-link-op $n1 $n2 orient left-up
$ns duplex-link-op $n2 $n3 orient up
$ns duplex-link-op $n3 $n4 orient right-up
$ns duplex-link-op $n4 $n5 orient right
$ns duplex-link-op $n5 $n6 orient right-down
$ns duplex-link-op $n6 $n7 orient down
$ns duplex-link-op $n7 $n8 orient left-down
$ns duplex-link-op $n8 $n1 orient left

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

#Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Create a Null agent (a traffic sink) and attach it to node n4
set null0 [new Agent/Null]
$ns attach-agent $n4 $null0

#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0
```

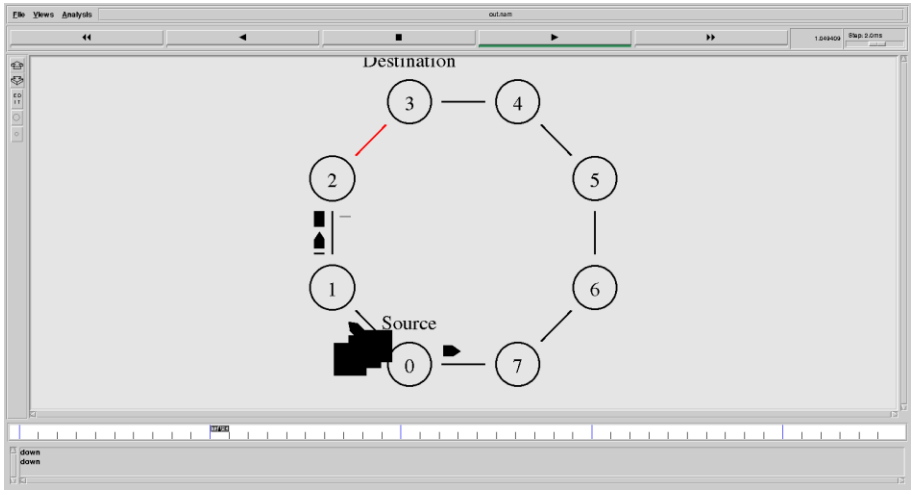
```
#Schedule events for the CBR agent and the network dynamics
$ns at 0.0 "$n1 label Source"
$ns at 0.0 "$n4 label Destination"
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n3 $n4
$ns rtmodel-at 2.0 up $n3 $n4
$ns at 4.5 "$cbr0 stop"

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

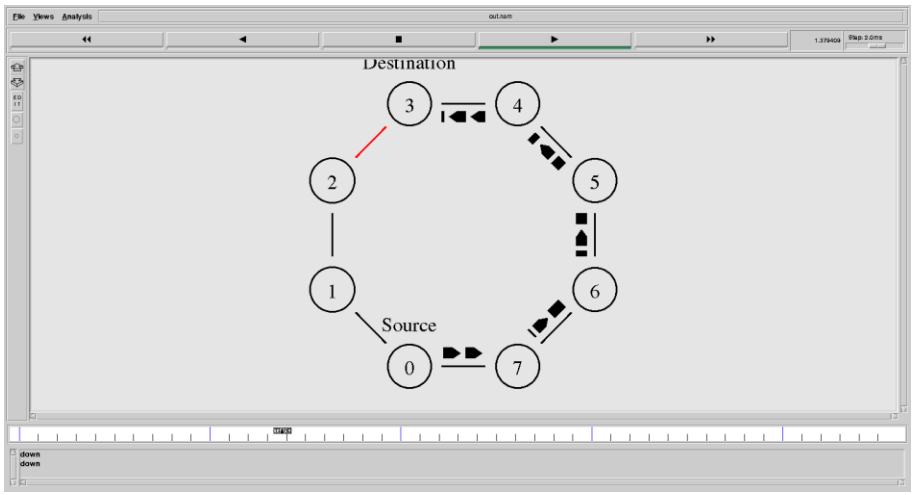
Output

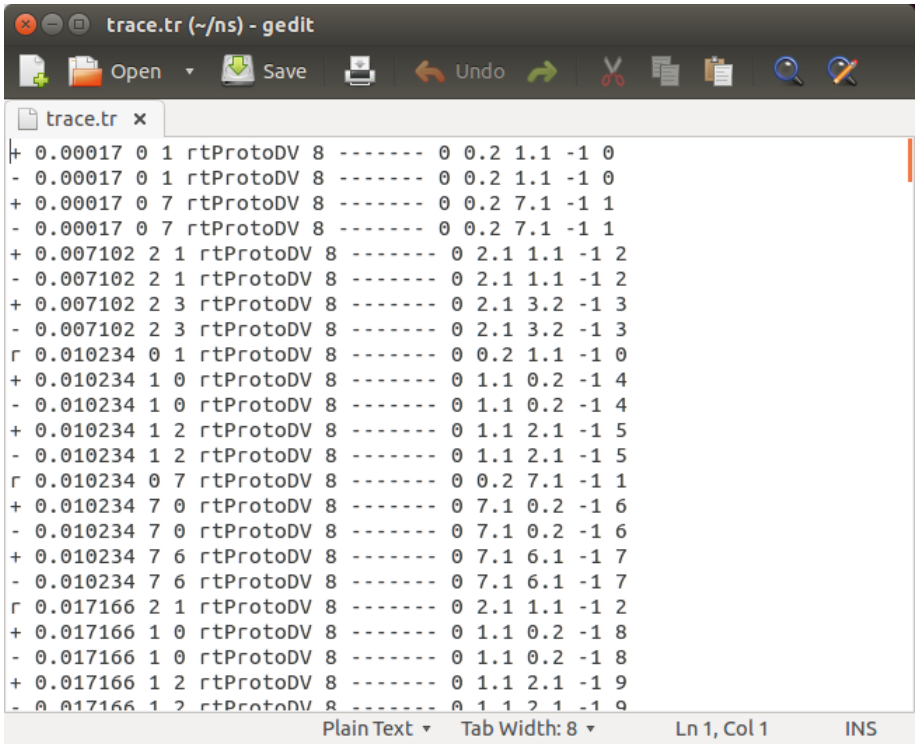
$ ns distvect.tcl











Result

Thus, the simulation for Distance vector and link state routing protocols was done using NS2.

Ex. No: 10	Simulation of Transport layer Protocols and analysis of congestion control techniques in the network

Aim

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

Procedure

Network Simulator (NS2)

Congestion Control Algorithms

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

Case Study: A simple Wireless network.

Ad hoc routing, mobile IP, sensor-MAC Tracing, visualization, and various utilities NS (Network Simulators).

Most of the commercial simulators are GUI driven, while some network simulators are CLI driven. The network model / configuration describes the state of the network (nodes, routers, switches, links) and the events (data transmissions, packet error etc.). An important output of simulations are the trace files. Trace files log every packet, every event that occurred in the simulation and are used for analysis. Network simulators can also provide other tools to facilitate visual analysis of trends and potential trouble spots.

Most network simulators use discrete event simulation, in which a list of pending "events" is stored, and those events are processed in order, with some events triggering future events—such as the event of the arrival of a packet at one node triggering the event of the arrival of that packet at a downstream node.

Simulation of networks is a very complex task. For example, if congestion is high, then estimation of the average occupancy is challenging because of high variance. To estimate the likelihood of a buffer overflow in a network, the time required for an accurate answer can be extremely large.

Specialized techniques such as "control variates" and "importance sampling" have been developed to speed simulation.

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)

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, analyse 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.

Packet loss

Packet loss occurs when one or more packets of data travelling across a computer network fail to reach their destination. Packet loss is distinguished as one of the three main error types encountered in digital communications; the other two being bit error and spurious packets caused due to noise.

Packets can be lost in a network because they may be dropped when a queue in the network node overflows. The amount of packet loss during the steady state is another important property of a congestion control scheme. The larger the value of packet loss, the more difficult it is for transport layer protocols to maintain high bandwidths, the sensitivity to loss of individual packets, as well as to frequency and patterns of loss among longer packet sequences is strongly dependent on the application itself.

Throughput

Throughput is the main performance measure characteristic, and most widely used. In communication networks, such as Ethernet or packet radio, throughput or network throughput is the average rate of successful message delivery over a communication channel. Throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second or data packets per time slot. This measures how soon the receiver can get a certain amount of data send by the sender. It is determined as the ratio of the total data received to the end-to-end delay. Throughput is an important factor which directly impacts the network performance.

Delay

Delay is the time elapsed while a packet travels from one point e.g., source premise or network ingress to destination premise or network degrees. The larger the value of delay, the more difficult it is for transport layer protocols to maintain high bandwidths. We will calculate end to end delay

Queue Length

A queuing system in networks can be described as packets arriving for service, waiting for service if it is not immediate, and if having waited for service, leaving the system after being served. Thus, queue length is very important characteristic to determine that how well the active queue management of the congestion control algorithm has been working.

Congestion control Algorithms

Slow-start is used in conjunction with other algorithms to avoid sending more data than the network is capable of transmitting, that is, to avoid causing network congestion. The additive increase/multiplicative decrease (AIMD) algorithm is a feedback control algorithm. AIMD combines linear growth of the congestion window with an exponential reduction when a congestion takes place. Multiple flows using AIMD congestion control will eventually converge to use equal amounts of a contended link. Fast Retransmit is an enhancement to TCP that reduces the time a sender waits before retransmitting a lost segment.

Program

```
include <wifi_lte/wifi_lte_rtable.h>
struct r_hist_entry *elm, *elm2;
int num_later = 1;
elm = STAILQ_FIRST(&r_hist_);
while (elm != NULL && num_later <= num_dup_acks_)
{
        num_later;
        elm = STAILQ_NEXT(elm, linfo_);
}
if (elm != NULL)
{
        elm = findDataPacketInRecvHistory(STAILQ_NEXT(elm,linfo_));
        if (elm != NULL)
        {
                elm2 = STAILQ_NEXT(elm, linfo_);
                while(elm2 != NULL)
                {
                        if (elm2->seq_num_ < seq_num && elm2->t_recv_ < time)
                        {
                                STAILQ_REMOVE(&r_hist_,elm2,r_hist_entry,linfo_);
                                delete elm2;
                        }
                        else
                        {
                                elm = elm2;
                                elm2 = STAILQ_NEXT(elm, linfo_);
                        }
```

```
                }
        }
}
void DCCPTFRCAgent::removeAcksRecvHistory()
{
        struct r_hist_entry *elm1 = STAILQ_FIRST(&r_hist_);
        struct r_hist_entry *elm2;
        int num_later = 1;
        while (elm1 != NULL && num_later <= num_dup_acks_)
        {
                num_later;
                elm1 = STAILQ_NEXT(elm1, linfo_);
        }
        if(elm1 == NULL)
        return;
        elm2 = STAILQ_NEXT(elm1, linfo_);
        while(elm2 != NULL)
        {
                if (elm2->type_ == DCCP_ACK)
                {
                        STAILQ_REMOVE(&r_hist_,elm2,r_hist_entry,linfo_);
                        delete elm2;
                }
                else
                {
                        elm1 = elm2;
                }
                elm2 = STAILQ_NEXT(elm1, linfo_);
        }
}
inline r_hist_entry *DCCPTFRCAgent::findDataPacketInRecvHistory(r_hist_entry *start)
{
        while(start != NULL && start->type_ == DCCP_ACK)
        start = STAILQ_NEXT(start,linfo_);
        return start;
}
```

Result

Thus, we have Studied Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.