JERUSALEM COLLEGE OF ENGINEERING

(An Autonomous Institution)
(Approved by AICTE, Affiliated to Anna University,
Accredited by NBA and NAAC with 'A' Grade)
Velachery Main Road, Narayanapuram,Pallikaranai, Chennai-600100

NETWORKS LABORATORY [JCS1411]

RECORD NOTEBOOK

ACADEMIC YEAR: 2022 - 2023

II YEAR / IV SEM

REGULATION: 2019

DEPARTMENT OF INFORMATION TECHNOLOGY

VISION OF THE DEPARTMENT

Department of Information Technology strives to provide quality education, academic excellence based on ethical and societal values, exposing students to all concepts, so as to promote global competitiveness in higher education, multi-disciplinary research and entrepreneurship.

MISSION OF THE DEPARTMENT

- To attain academic excellence through innovative practices in teaching and research methodologies.
- To produce globally competent information technologists and entrepreneurs.
- To motivate students to pursue higher education interlaced with communication skills leading to lifelong learning and societal transformations.
- To provide excellence in multi-disciplinary research and development activities rooted in ethical and moral values.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: To ensure graduates will be proficient in utilizing the fundamental knowledge of basic sciences, mathematics and Information Technology for the applications relevant to variousstreams of Engineering and Technology.

PEO2: To enrich graduates with the core competencies necessary for applying knowledge of computers and telecommunications equipment to store, retrieve, transmit, manipulate and analyze data in the context of business enterprise.

PEO3: To enable graduates to think logically, pursue lifelong learning and will have the capacity to understand technical issues related to computing systems and to design optimal solutions.

PEO4: To enable graduates to develop hardware and software systems by understanding the importance of social, business and environmental needs in the human context.

PEO5: To enable graduates to gain employment in organizations and establish themselves as professionals by applying their technical skills to solve real world problems and meet the diversified needs of industry, academia and research.

PROGRAM SPECIFIC OBJECTIVES (PSOs)

PSO-I: Proficiency to effectively integrate IT-based solutions for contemporary cross-functional applications.

PSO-II: Ability to analyze, design, implement and evaluate the information systems with ethics, to meet the local and global requirements for scientific and industry solutions.

JERUSALEM COLLEGE OF ENGINEERING

(An Autonomous Institution)
(Approved by AICTE, Affiliated to Anna University,
Accredited by NBA and NAAC with ‘A’ Grade)
Velachery Main Road, Narayanapuram,Pallikaranai, Chennai-600100

Name…………………………………………………………………………………

Year…………………………………**Semester**………………**Branch**……………….

Regulation:…………………………

Register No.

Certified that this is a Bonafide Record work done by the above student in the ……………….. …..……………………………………………………Laboratory during the year 20 - 20 .

----------------------- -------------------------

Signature of Lab. In charge **Signature of Head of the Dept.**

<u>EXAMINERS</u>

DATE:_______________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

SYLLABUS

OBJECTIVES:

- To learn Socket programming
- To study various networking commands
- To implement and analyze various network protocols
- To learn and implement various socket programming concepts
- To simulate and analyze the performance of various network protocols

S. No.	NAME OF EXPERIMENT
1	Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture pingand traceroute PDUs using a network protocol analyzer and examine
2	Write a code simulating ARP /RARP protocols.
3	Write an application that draws basic graphical primitives on the screen. Write a socket program for HTTP web page upload and download.
4	Applications using TCP Sockets like Echo client and echo server
5	Applications using TCP Sockets like Chat
6	Applications using TCP Sockets like File Transfer
7	Applications of DNS using UDP Sockets
8	Study of Network simulator and Simulation of Congestion Control Algorithms
9	Study of TCP/UDP performance using Simulation tool
10	Simulation of Distance Vector/ Link State Routing algorithm.
11	Performance evaluation of Routing protocols using Simulation tool.
12	Simulation of error correction code (like CRC).

LIST OF EXPERIMENTS

INDEX

S.No	**Date**	**Name of the Experiment**	**Page No.**	**Marks**	**Signature with Date**

Average Marks:
Signature:

EX NO 1 **Study of basic Network commands**

Aim:

Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute.

tcpdump

Description:

tcpdump prints out a description of the contents of packets on a network interface that match the boolean expression; the description is preceded by a time stamp, printed, by default, as hours, minutes, seconds, and fractions of a second since midnight.

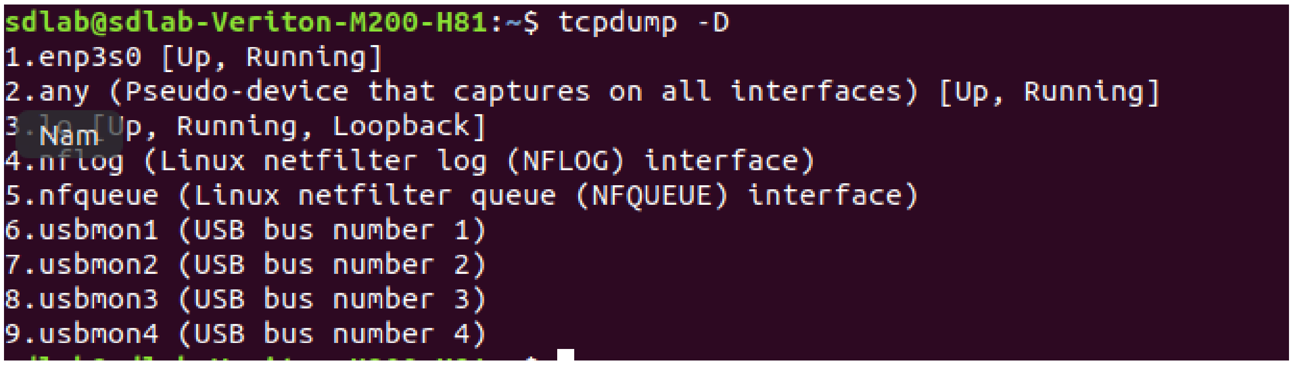
It can also be run with the -w flag, which causes it to save the packet data to a file for later analysis, and/or with the -r flag, which causes it to read from a saved packet file rather than to read packets from a network interface (please note tcpdump is protected via an enforcing apparmor(7) profile in Ubuntu which limits the files tcpdump may access).

It can also be run with the -V flag,which causes it to read a list of saved packet files. In all cases, only packets that match expression will be processed by tcpdump.

INPUT:**tcpdump -D**

To list number of available interfaces on the system, run the following command with -D option.

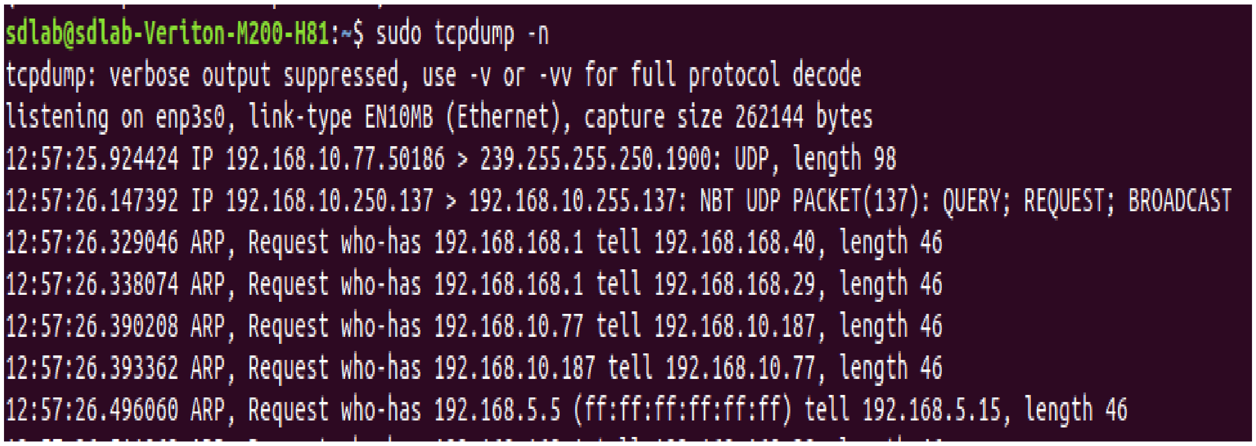
OUTPUT:



```
sdlab@sdlab-Veriton-M200-H81:~$ tcpdump -D
1.enp3s0 [Up, Running]
2.any (Pseudo-device that captures on all interfaces) [Up, Running]
3.lo [Up, Running, Loopback]
4.nflog (Linux netfilter log (NFLOG) interface)
5.nfqueue (Linux netfilter queue (NFQUEUE) interface)
6.usbmon1 (USB bus number 1)
7.usbmon2 (USB bus number 2)
8.usbmon3 (USB bus number 3)
9.usbmon4 (USB bus number 4)
```

INPUT : tcpdump -n

OUTPUT :



```
sdlab@sdlab-Veriton-M200-H81:~$ sudo tcpdump -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp3s0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:57:25.924424 IP 192.168.10.77.50186 > 239.255.255.250.1900: UDP, length 98
12:57:26.147392 IP 192.168.10.250.137 > 192.168.10.255.137: NBT UDP PACKET(137): QUERY; REQUEST; BROADCAST
12:57:26.329046 ARP, Request who-has 192.168.168.1 tell 192.168.168.40, length 46
12:57:26.338074 ARP, Request who-has 192.168.168.1 tell 192.168.168.29, length 46
12:57:26.390208 ARP, Request who-has 192.168.10.77 tell 192.168.10.187, length 46
12:57:26.393362 ARP, Request who-has 192.168.10.187 tell 192.168.10.77, length 46
12:57:26.496060 ARP, Request who-has 192.168.5.5 (ff:ff:ff:ff:ff:ff) tell 192.168.5.15, length 46
```

Writing to a file

```
# tcpdump -w /path/to/file

# tcpdump -w /var/tmp/tcpdata.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
1 packet captured
2 packets received by filter
0 packets dropped by kernel
```

Reading from a file

```
# tcpdump -r /path/to/file

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
16:15:05.051896 IP blog.ssh > 10.0.3.1.32855: Flags [P.], seq 2546456553:2546456749, ack
1824683693, win 355, options [nop,nop,TS val 620879437 ecr 620879348], length 196
```

Specifying the number of packets to capture

```
# tcpdump -c 10

# tcpdump host google.com
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C
0 packets captured
4 packets received by filter
0 packets dropped by kernel
```

Netstat:

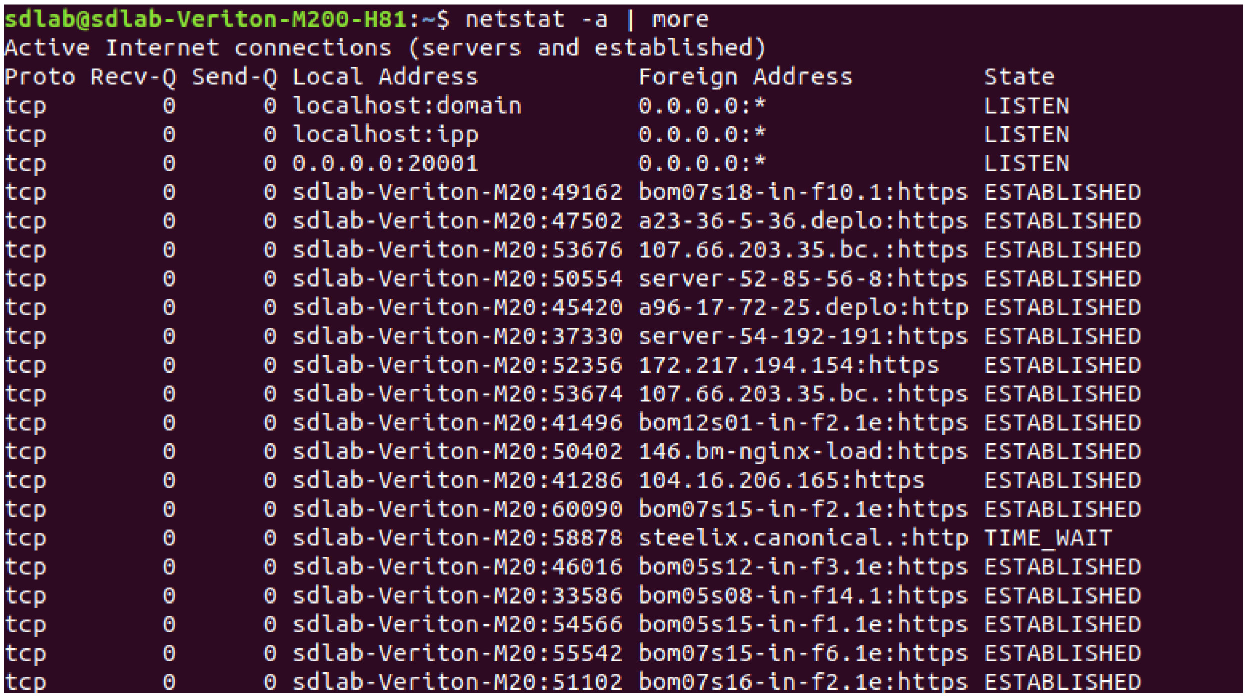
Description :

Netstat prints information about the Linux networking subsystem. The type of information printed is controlled by the first argument, as follows:

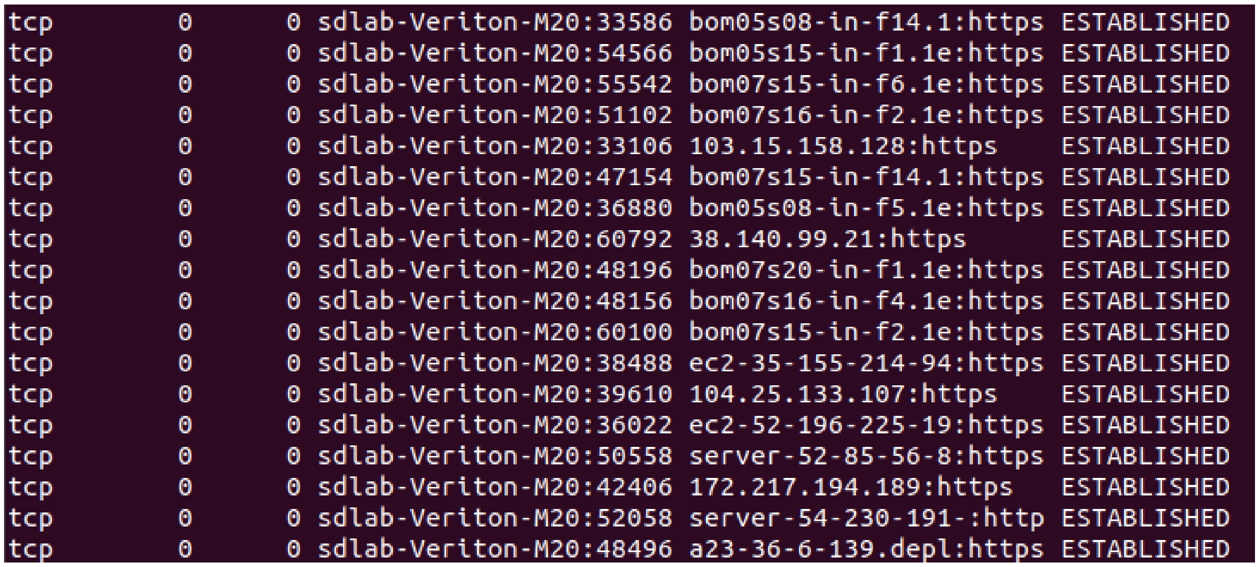
By default, netstat displays a list of open sockets. If you don't specify any address families, then the active sockets of all configured address families will be printed.

Input :**netstat -a | more**

To show both listening and non-listening sockets.



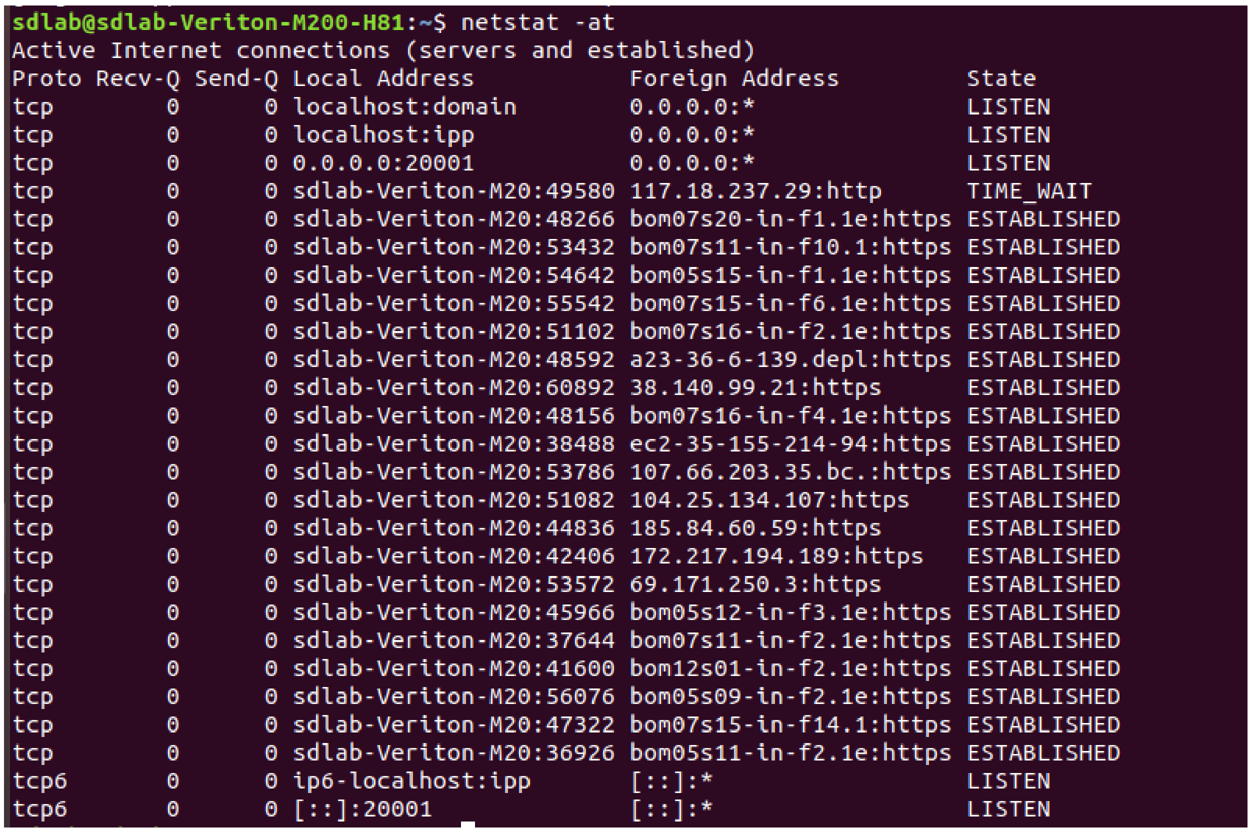
```
sdlab@sdlab-Veriton-M200-H81:~$ netstat -a | more
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:20001           0.0.0.0:*               LISTEN
tcp        0      0 sdlab-Veriton-M20:49162 bom07s18-in-f10.1:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:47502 a23-36-5-36.deplo:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:53676 107.66.203.35.bc.:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:50554 server-52-85-56-8:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:45420 a96-17-72-25.deplo:http ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:37330 server-54-192-191:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:52356 172.217.194.154:https   ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:53674 107.66.203.35.bc.:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:41496 bom12s01-in-f2.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:50402 146.bm-nginx-load:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:41286 104.16.206.165:https    ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:60090 bom07s15-in-f2.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:58878 steelix.canonical.:http TIME_WAIT
tcp        0      0 sdlab-Veriton-M20:46016 bom05s12-in-f3.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:33586 bom05s08-in-f14.1:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:54566 bom05s15-in-f1.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:55542 bom07s15-in-f6.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:51102 bom07s16-in-f2.1e:https ESTABLISHED
```



```
tcp        0      0 sdlab-Veriton-M20:33586 bom05s08-in-f14.1:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:54566 bom05s15-in-f1.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:55542 bom07s15-in-f6.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:51102 bom07s16-in-f2.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:33106 103.15.158.128:https    ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:47154 bom07s15-in-f14.1:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:36880 bom05s08-in-f5.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:60792 38.140.99.21:https      ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:48196 bom07s20-in-f1.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:48156 bom07s16-in-f4.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:60100 bom07s15-in-f2.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:38488 ec2-35-155-214-94:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:39610 104.25.133.107:https    ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:36022 ec2-52-196-225-19:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:50558 server-52-85-56-8:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:42406 172.217.194.189:https   ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:52058 server-54-230-191-:http ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:48496 a23-36-6-139.depl:https ESTABLISHED
```

Input :**netstat -at**

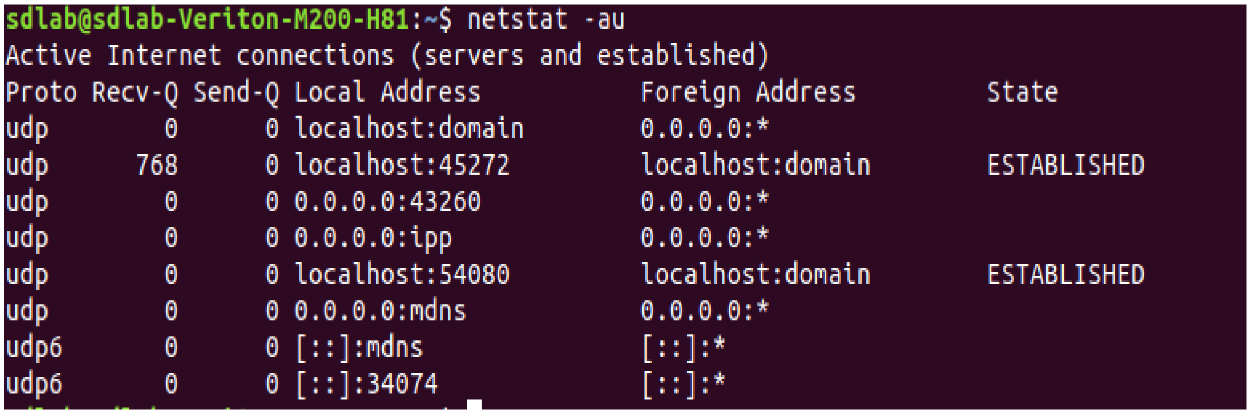
To list all tcp ports.



```
sdlab@sdlab-Veriton-M200-H81:~$ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:domain        0.0.0.0:*               LISTEN
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:20001           0.0.0.0:*               LISTEN
tcp        0      0 sdlab-Veriton-M20:49580 117.18.237.29:http      TIME_WAIT
tcp        0      0 sdlab-Veriton-M20:48266 bom07s20-in-f1.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:53432 bom07s11-in-f10.1:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:54642 bom05s15-in-f1.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:55542 bom07s15-in-f6.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:51102 bom07s16-in-f2.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:48592 a23-36-6-139.depl:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:60892 38.140.99.21:https      ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:48156 bom07s16-in-f4.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:38488 ec2-35-155-214-94:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:53786 107.66.203.35.bc.:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:51082 104.25.134.107:https    ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:44836 185.84.60.59:https      ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:42406 172.217.194.189:https   ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:53572 69.171.250.3:https      ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:45966 bom05s12-in-f3.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:37644 bom07s11-in-f2.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:41600 bom12s01-in-f2.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:56076 bom05s09-in-f2.1e:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:47322 bom07s15-in-f14.1:https ESTABLISHED
tcp        0      0 sdlab-Veriton-M20:36926 bom05s11-in-f2.1e:https ESTABLISHED
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN
tcp6       0      0 [::]:20001              [::]:*                  LISTEN
```

Input :**netstat -au**

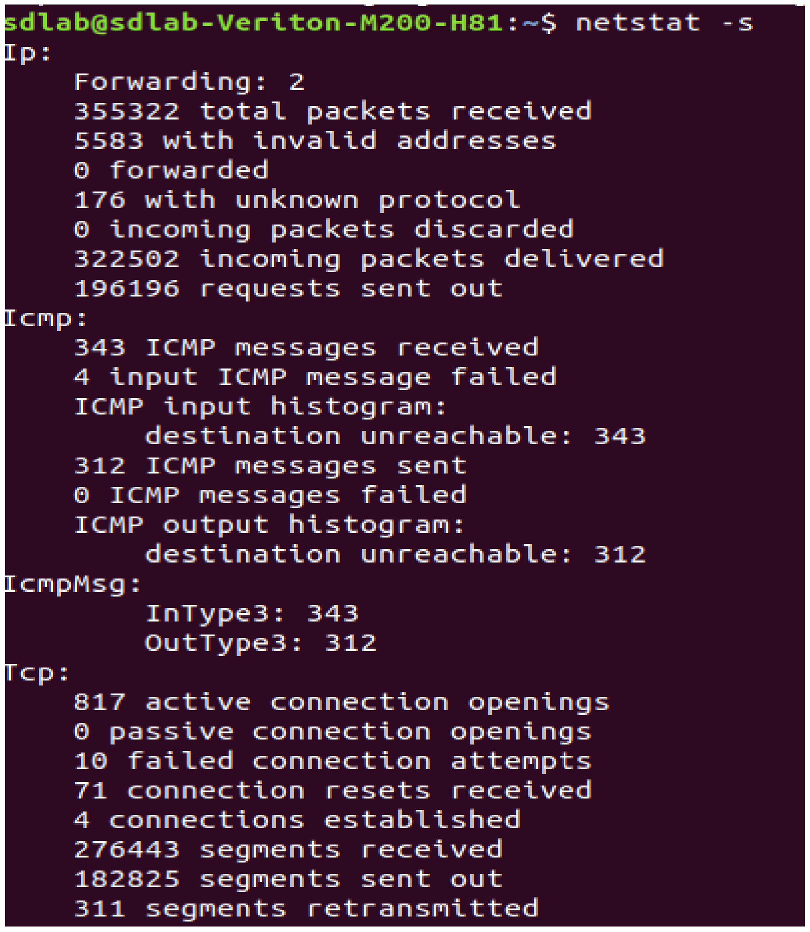
To list all udp ports.



```
sdlab@sdlab-Veriton-M200-H81:~$ netstat -au
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 localhost:domain        0.0.0.0:*
udp      768      0 localhost:45272         localhost:domain        ESTABLISHED
udp        0      0 0.0.0.0:43260           0.0.0.0:*
udp        0      0 0.0.0.0:ipp             0.0.0.0:*
udp        0      0 localhost:54080         localhost:domain        ESTABLISHED
udp        0      0 0.0.0.0:mdns            0.0.0.0:*
udp6       0      0 [::]:mdns               [::]:*
udp6       0      0 [::]:34074              [::]:*
```

Input :**netstat -s**

To list the statistics for all ports.



```
sdlab@sdlab-Veriton-M200-H81:~$ netstat -s
Ip:
    Forwarding: 2
    355322 total packets received
    5583 with invalid addresses
    0 forwarded
    176 with unknown protocol
    0 incoming packets discarded
    322502 incoming packets delivered
    196196 requests sent out
Icmp:
    343 ICMP messages received
    4 input ICMP message failed
    ICMP input histogram:
        destination unreachable: 343
    312 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 312
IcmpMsg:
        InType3: 343
        OutType3: 312
Tcp:
    817 active connection openings
    0 passive connection openings
    10 failed connection attempts
    71 connection resets received
    4 connections established
    276443 segments received
    182825 segments sent out
    311 segments retransmitted
```

ifconfig

Description:

Ifconfig is used to configure the kernel-resident network interfaces. It is used at boot time to set up

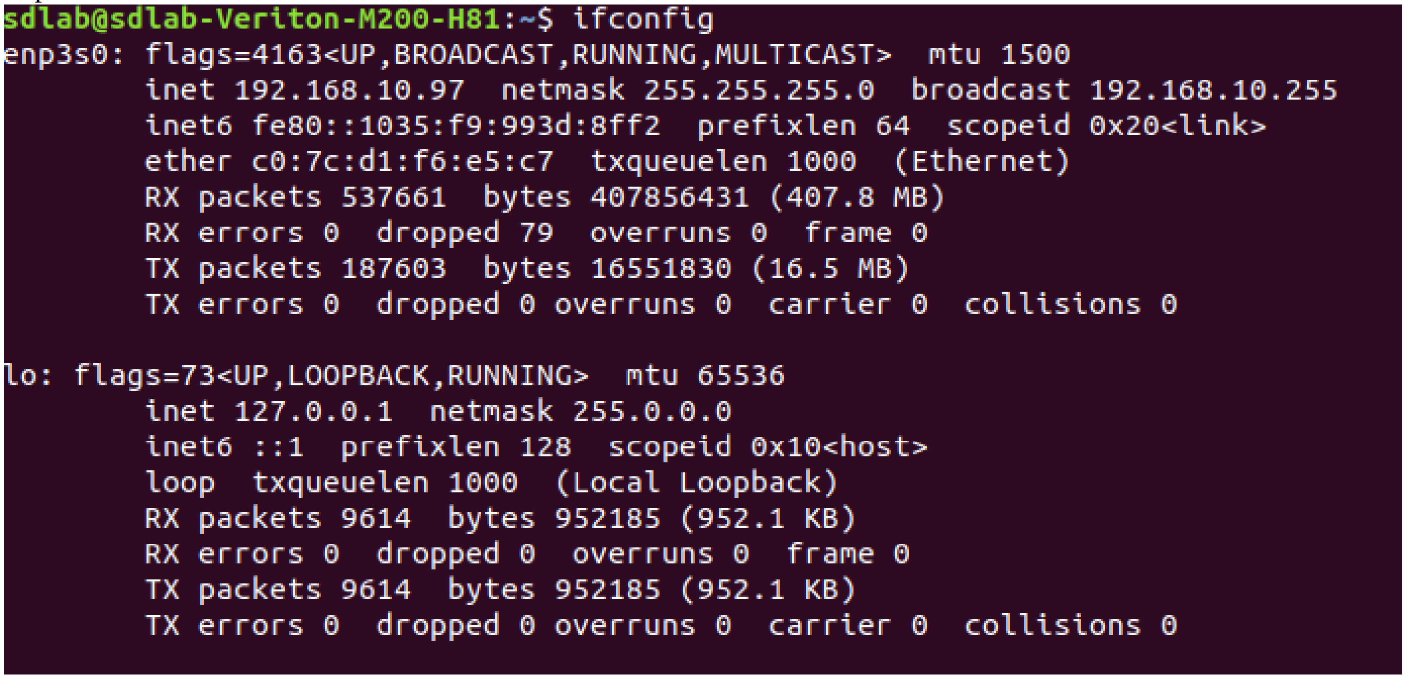
interfaces as necessary. After that, it is usually only needed when debugging or when system tuning is needed.

If no arguments are given, ifconfig displays the status of the currently active interfaces. If a single

interface argument is given, it displays the status of the given interface only; if a single -a argument is given, it displays the status of all interfaces, even those that are down. Otherwise, it configures an interface.

Input: ifconfig

Output:



```
sdlab@sdlab-Veriton-M200-H81:~$ ifconfig
enp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.10.97  netmask 255.255.255.0  broadcast 192.168.10.255
        inet6 fe80::1035:f9:993d:8ff2  prefixlen 64  scopeid 0x20<link>
        ether c0:7c:d1:f6:e5:c7  txqueuelen 1000  (Ethernet)
        RX packets 537661  bytes 407856431 (407.8 MB)
        RX errors 0  dropped 79  overruns 0  frame 0
        TX packets 187603  bytes 16551830 (16.5 MB)
        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 9614  bytes 952185 (952.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9614  bytes 952185 (952.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

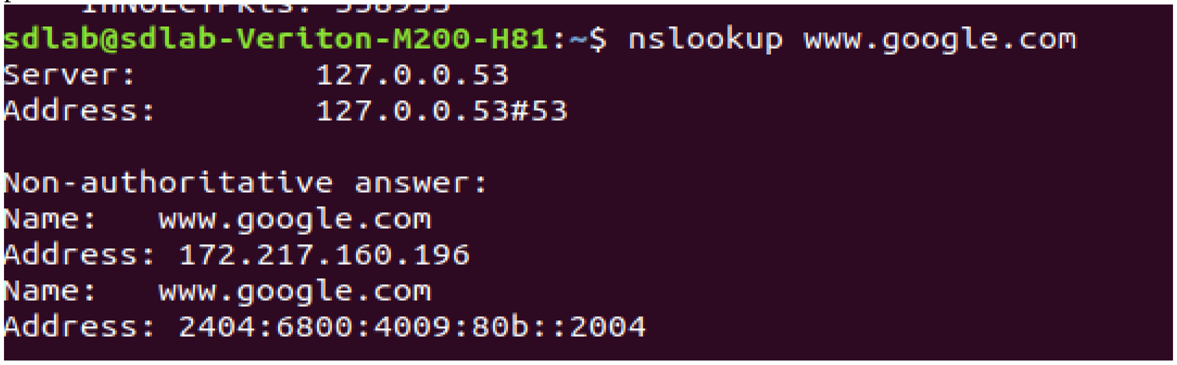
nslookup

Description:

Nslookup is a program to query Internet domain name servers. Nslookup has two modes: interactive and non-interactive. Interactive mode allows the user to query name servers for information about various hosts and domains or to print a list of hosts in a domain. Non-interactive mode is used to print just the name and requested information for a host or domain.

Input: nslookup www.google.com

Output:



```
sdlab@sdlab-Veriton-M200-H81:~$ nslookup www.google.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   www.google.com
Address: 172.217.160.196
Name:   www.google.com
Address: 2404:6800:4009:80b::2004
```

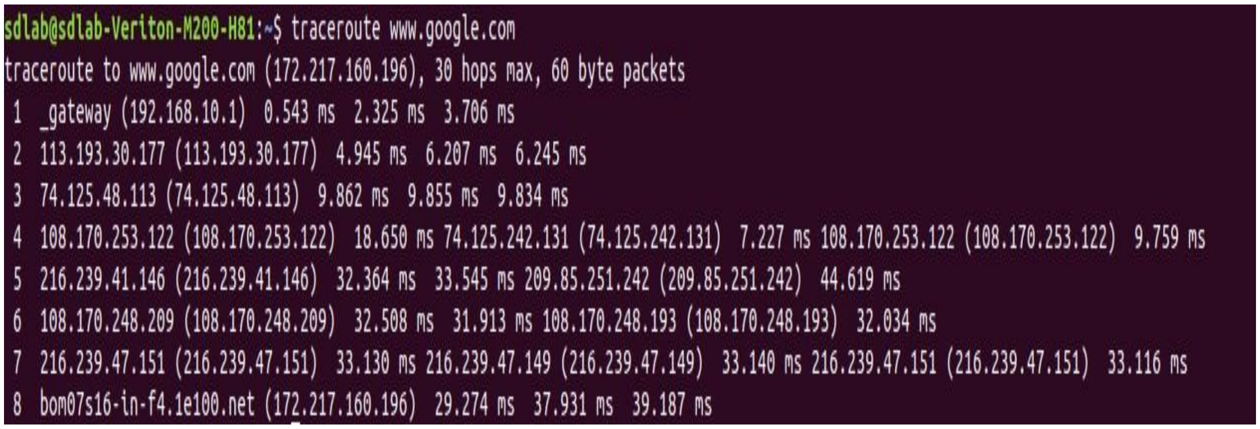
traceroute

Description:

traceroute tracks the route packets taken from an IP network on their way to a given host. It utilizes the IP protocol's time to live (TTL) field and attempts to elicit an ICMP TIME_EXCEEDED response from each gateway along the path to the host.

Input: traceroute www.google.com

Output :



```
sdlab@sdlab-Veriton-M200-H81:~$ traceroute www.google.com
traceroute to www.google.com (172.217.160.196), 30 hops max, 60 byte packets
 1  _gateway (192.168.10.1)  0.543 ms  2.325 ms  3.706 ms
 2  113.193.30.177 (113.193.30.177)  4.945 ms  6.207 ms  6.245 ms
 3  74.125.48.113 (74.125.48.113)  9.862 ms  9.855 ms  9.834 ms
 4  108.170.253.122 (108.170.253.122)  18.650 ms 74.125.242.131 (74.125.242.131)  7.227 ms 108.170.253.122 (108.170.253.122)  9.759 ms
 5  216.239.41.146 (216.239.41.146)  32.364 ms  33.545 ms 209.85.251.242 (209.85.251.242)  44.619 ms
 6  108.170.248.209 (108.170.248.209)  32.508 ms  31.913 ms 108.170.248.193 (108.170.248.193)  32.034 ms
 7  216.239.47.151 (216.239.47.151)  33.130 ms 216.239.47.149 (216.239.47.149)  33.140 ms 216.239.47.151 (216.239.47.151)  33.116 ms
 8  bom07s16-in-f4.1e100.net (172.217.160.196)  29.274 ms  37.931 ms  39.187 ms
```

RESULT:

EX NO 2A **Simulating PING Commands**

AIM:
To write the java program for simulating ping command.

ALGORITHM:

1. Start the program.
2. Include necessary package in java.
3. To create a process object p to implement the ping command.
4. Declare one BufferedReader stream class object.
5. Get thedetails of the server
   5.1: length of the IP address.
   5.2: time required to get the details.
   5.3: send packets, receive packets and lost packets.
   5.4: minimum, maximum and average times.
6. Print the results.
7. Stop the program.

PROGRAM:

```
package ping;
import java.io.*;
import java.net.*;
public class Ping
{
public static void main(String[] args)
{
try
{
String str;
System.out.print(" Enter the IP Address to be Ping : ");
BufferedReader buf1=new BufferedReader(new  InputStreamReader(System.in));
String ip=buf1.readLine();
Runtime H=Runtime.getRuntime();
Process p=H.exec("ping " + ip);
InputStream in=p.getInputStream();
BufferedReader buf2=new BufferedReader(new
InputStreamReader(in));
while((str=buf2.readLine())!=null)
{
System.out.println(" " + str);
}
}
catch(Exception e)
{
System.out.println(e.getMessage());
}
}
}
```

OUTPUT:

```
 Enter the IP Address to be Ping : 10.10.30.47
PING 10.10.30.47 (10.10.30.47) 56(84) bytes of data.
 64 bytes from 10.10.30.47: icmp_seq=1 ttl=128 time=2.24 ms
 64 bytes from 10.10.30.47: icmp_seq=2 ttl=128 time=1.96 ms
 64 bytes from 10.10.30.47: icmp_seq=3 ttl=128 time=0.193 ms
 64 bytes from 10.10.30.47: icmp_seq=4 ttl=128 time=0.199 ms
 64 bytes from 10.10.30.47: icmp_seq=5 ttl=128 time=0.524 ms
```

RESULT:

EX NO 2B **Simulating traceroute command**

AIM: To write a program to simulate Traceroute command.

ALGORITHM:

1. Start the program and declare the variables.
2. Create the document file path.txt and give the all details
3. Open the document file path.txt and compare the input with the details in the path.txt
4. Trace the route
5. Stop the program

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int main()
        {
                 char ip1[25],ip2[25],ip3[25],ip4[25],ip5[25];
                char destn[25];
                 FILE *fp;
                printf("\n tracerroute");
                scanf("%s",&destn);
                fp=fopen("path.txt","r");
                while(!feof(fp))
                 {
                        fscanf(fp,"%s\t\t%s\t\t%s\t\t%s\t\t%s\n",&ip1,&ip2,&ip3,&ip4,&ip5);
                         if((strcmp(destn,ip4)==0)||(strcmp(destn,ip5)==0))
                         {
                                printf("\n tracing route to %s\n over a maximum of    30hops",ip4);
                                printf("\n]%s\n2]%s[%s]\n",ip2,ip3,ip4,ip5);
                                printf("\n trace complete");
                                exit(0);
                         }
                 }
                return 0;
        }
```

OUTPUT:

path.txt

```
3.21.191.19       LocalGateway[67.195.160.76]  145.42.22.125 125.22.42.145 www.rediff.com
3.21.191.19       LocalGateway[67.195.160.76]  213.36.144.59 59.144.36.215 www.monster.com
3.21.191.19       LocalGateway[67.195.160.76] 216.115.96.52 76.13.0.191 www.facebook.com
tracerroute www.rediff.com
tracing route to 125.22.42.145
over a maximum of    30hops
]LocalGateway[67.195.160.76]
2]145.42.22.125[125.22.42.145]
trace complete
```

RESULT:

EX NO 3 Create a socket for http for web page upload and download

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

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:
webpageclient.java

```
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",4015);
System.out.println("Client is running. ");
try
   {
System.out.println("Reading image from disk. ");
img = ImageIO.read(new File("/home/s/image/karuppu.jpeg"));
     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();
  }
}
```

webpageserver.java

```
package  web;
import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;
public class Server
{
public static void main(String args[]) throws Exception
    {
     ServerSocket server=null;
     Socket socket;
server=new ServerSocket(4015);
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);
  }
}
```

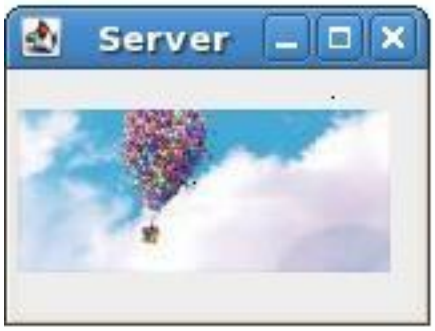
OUTPUT:

CLIENT SIDE

```
Client is running.
Reading image from disk.
Sending image to server.
Image sent to server.
```

SERVER SIDE

```
Server Waiting for image
Client connected.
Image Size: 2KB
```



RESULT:

EX NO 4A Applications using TCP sockets like echo client and echo server

AIM :

To write a program in C to implement TCP Echo Client Server (Iterative model).

ALGORITHM :

Server

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

Client:

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

PROGRAM

SERVER

```
//iterserv.c -- Iterative Echo Server

#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<string.h>
#include<unistd.h>
#define PORT 7676

int main(int argc, char *argv[])

{

char buffer[20];

int sockfd,connfd,a,len;
struct sockaddr_in servaddr,cliaddr;
sockfd=socket(AF_INET,SOCK_STREAM,0);

if(sockfd==-1)
        printf("ERROR CREATING SOCKET!");

bzero(&servaddr,sizeof(servaddr));

servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(PORT);
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);

bind(sockfd,(struct sockaddr *)&servaddr,sizeof(servaddr));

if((a=listen(sockfd,5))<0)
        printf("ERROR IN LISTEN FUNCTION!");
while(1)
        {
        len=sizeof(cliaddr);
        connfd=accept(sockfd,(struct sockaddr*)&cliaddr,&len);
        strcpy(buffer,"");
        read(connfd,buffer,10);
        printf("Message Received and Echoed : %s",buffer);
        write(connfd,buffer,sizeof(buffer));
        }
close(sockfd);
}
```

CLIENT

```
//itercli.c -- Echo client
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<string.h>
#include<unistd.h>

#define PORT 7676

int main(int argc,char *argv[])
{
int sockfd;
struct sockaddr_in serv;
char buff[20];

sockfd=socket(AF_INET,SOCK_STREAM,0);

memset(&serv,0,sizeof(serv));


serv.sin_family=AF_INET;

serv.sin_port=htons(PORT);

serv.sin_addr.s_addr=inet_addr(argv[1]);


if(connect(sockfd,(struct sockaddr *)&serv,sizeof(serv))<0)

        printf("ERROR IN CONNECT");


printf("ENTER THE STRING TO ECHO :");

fgets(buff,sizeof(buff),stdin);
write(sockfd,buff,sizeof(buff));
strcpy(buff," ");
read(sockfd,buff,sizeof(buff));

fputs(buff,stdout);

close(sockfd);

return;
}
```

//OUTPUT:

//SERVER

```
[Localhost@Tamil]$ cc iterserv.c
[Localhost@Tamil]$  ./a.out
Message Received and Echoed : Good
```

//CLIENT

```
[Localhost@Tamil]$ cc itercli.c
[Localhost@Tamil]$  ./a.out 127.0.0.1
ENTER THE STRING TO ECHO :Good Morning
Good Morning
[Localhost@Tamil]$
```

Result:

EX NO 4B Applications using TCP sockets like chat

AIM :

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

ALGORITHM :

Server

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

Client:

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

PROGRAM

SERVER

```
//tcpchatser.c -- TCP CHAT SERVER

#include<stdio.h>

#include<sys/types.h>
```

```
#include<netinet/in.h>

#include<sys/socket.h>

#include<string.h>

#include<unistd.h>

#include<stdlib.h>

#define PORT 4771


int main(int argc,char *argv[])

{

char buffer[100];

int sockfd,a,connfd,len,i=0;

pid_t pid;

struct sockaddr_in servaddr,cliaddr;

sockfd=socket(AF_INET,SOCK_STREAM,0);

if(sockfd==-1)

        {

        printf("Error creating socket\n");

        exit(0);

        }

printf("Server Socket Created Successfully.\n");

bzero((struct sockaddr *)&servaddr,sizeof(servaddr));

servaddr.sin_family=AF_INET;

servaddr.sin_port=htons(PORT);

servaddr.sin_addr.s_addr=htonl(INADDR_ANY);


if(bind(sockfd,(struct sockaddr *)&servaddr,sizeof(servaddr))<0)

        {
```

```
		printf("Error in BIND function");
		exit(0);
		}
	printf("Server Socket Binded.\n");
	if((a=listen(sockfd,5))==-1)
		{
		printf("Error in LISTEN function");
		exit(0);
		}
	printf("Server Socket Listened...\n");
	len=sizeof(cliaddr);
	if((connfd=accept(sockfd,(struct sockaddr *)&cliaddr,&len))<0)
		{
		printf("Error in ACCEPT");
		exit(0);
		}
	do
		{
		strcpy(buffer," ");
		read(connfd,buffer,100);
		printf("From client :%s",buffer);
		if(strcmp(buffer,"bye\n")==0)
			{
			printf("from Client: %s",buffer);
			goto stop;
			}
		printf("Server :");
```

```
        fgets(buffer,sizeof(buffer),stdin);

        write(connfd,buffer,100);

        }while(strcmp(buffer,"bye\n")!=0);

stop:

exit(0);


close(connfd);

return 0;

}
```

CLIENT

```
//tcpchatcli.c -- TCP CHAT CLIENT
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<string.h>
#include<unistd.h>
#include<stdlib.h>
#define PORT 4771


int main(int argc,char *argv[])

{

int sockfd;

struct sockaddr_in serv;

char buff[100];

sockfd=socket(AF_INET,SOCK_STREAM,0);

printf("Client Socket Created Successfully.\n");

memset(&serv,0,sizeof(serv));

serv.sin_family=AF_INET;

serv.sin_port=htons(PORT);
serv.sin_addr.s_addr=inet_addr(argv[1]);
if (connect(sockfd,(struct sockaddr *)&serv, sizeof(serv))<0)
        {
        printf("error in connect");
        exit(0);
        }
printf("Client Socket with Server Successfully.\n");
do
        {
        strcpy(buff," ");
        printf("Client. :");
        fgets(buff,100,stdin);
        write(sockfd,buff,100);
        if(strcmp(buff,"bye\n")==0)
        {
                printf("Client: %s",buff);
                goto stop;
                }
        strcpy(buff," ");
        read(sockfd,buff,sizeof(buff));
        printf("From Server :%s\n",buff);
        }while(strcmp(buff,"bye\n")!=0);
stop:
exit(0);


close(sockfd);
return 0;
}
```

OUTPUT

//**Server**

```
[Localhost@Tamil]$  cd ..
[Localhost@Tamil]$cd  TCPCHAT
[Localhost@Tamil]$cc tcpchatser.c
[Localhost@Tamil]$./a.out
Server Socket Created Successfully.
Server Socket Binded.
Server Socket Listened...
From client :Hi
Server :Hello ..How are you?
From client :Fine. How Are You?
Server :Fine
From client :Bye
Server :Bye
[Localhost@Tamil]$
```

//Client

```
[Localhost@Tamil]$cc tcpchatcli.c
[Localhost@Tamil]$./a.out 127.0.0.1
Client Socket Created Successfully.
Client Socket with Server Successfully.
Client.:Hi
From Server :Hello ..How are you?
Client.:Fine. How Are You?
From Server :Fine
Client.:Bye
From Server :Bye
[Localhost@Tamil]$
```

Result:

EX NO 4C **Applications using TCP sockets like file transfer**

AIM: To write a C program to transfer file using TCP Sockets

ALGORITHM:

SERVER:

1. Start the program, declare variables
2. Create a socket using the socket structure socket(AF_INET, SOCK_STREAM,0)
3. Set the socket family, IP address and the port using the server address
4. Set the socket address of 8 bytes to zero using the memset() function
5. Bind and listen the socket structure
6. Accept the client connection using the socket descriptor and the server address
7. Get the file name to be transferred, the server search the file and send the file into the client and then close the connection
8. Compile and Execute the program

CLIENT

1. Start the program
2. Create a socket using socket(AF_INET, SOCK_STREAM,0)
3. Set the 8 bytes address to be zero using memset() function set the socket address family and port using serv addr
4. Establish connection with the server, read the filename to be retrieved from server
5. Receive the file from the server and read the contents of the received file

PROGRAM
SERVER

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<error.h>
#include<string.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/wait.h>
#include<signal.h>
#include<sys/socket.h>
#define MYPORT 7014
#define BACKLOG 10
int main(void)
{
char buf[100],fname[30]; int n,nbytes;
int sockfd,new_fd,size,des; struct sockaddr_in maddr; struct sockaddr_in taddr;
if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
perror("SOCKET");
exit(1);
}
maddr.sin_family=AF_INET; maddr.sin_port=htons(MYPORT); maddr.sin_addr.s_addr=INADDR_ANY;
memset(&(maddr.sin_zero),'\0',8);

if(bind(sockfd,(struct sockaddr*)&maddr,sizeof(struct sockaddr))==-1)
{
perror("BIND");
exit(1);
}
if(listen(sockfd,BACKLOG)==-1)
{
perror("LISTEN");
exit(1);
}
printf("FILE TRANSFER\n");
while(1)
{
size=sizeof(struct   sockaddr_in);
if((new_fd=accept(sockfd,(struct sockaddr*)&taddr,&size))==-1)
{
perror("ACCEPT");
continue;
}
if((nbytes=recv(new_fd,fname,114,0))==-1)
{
perror("ERROR IN RECEIVING\n"); exit(1);
}
        if((des=open(fname,0))==-1)
{
perror("ERROR");
exit(0);
}
while((n=read(des,buf,100))>0)
{
if(send(new_fd,buf,n,0)==-1) perror("ERROR IN SENDING\n");
}
printf("FILE IS SENT AND READ SUCCESSFULLY\n"); fflush(stdout);
close(des);
}
return 0;
}
```

CLIENT

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<errno.h>
#include<string.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<sys/wait.h>
#include<signal.h>
#define MYPORT 7014
int main(int argc,char *argv[])
{
```

```
char buf[100],fname[14]; int nbytes,sockfd;
struct sockaddr_in taddr;
if(argc!=2)
{
fprintf(stderr,"usage:ClientHost Name\n"); exit(1);
}
if((sockfd=socket(AF_INET,SOCK_STREAM,0))==-1)
{
perror("SOCKET");
exit(1);
}
taddr.sin_family=AF_INET; taddr.sin_port=htons(MYPORT);
taddr.sin_addr.s_addr=htonl(INADDR_ANY); memset(&(taddr.sin_zero),'\0',8);
if(connect(sockfd,(struct sockaddr *)&taddr,sizeof(struct sockaddr))==-1)
{
perror("CONNECTING ERROR");
exit(1);
}
fflush(stdout);
printf("FILE TRANSFER\n");
fflush(stdout); printf("INPUT\n");
printf("ENTER THE FILE NAME\n");
scanf("%s",fname);
if(send(sockfd,fname,14,0)==- 1)
perror("SENDING ERROR");
printf("OUTPUT\n");
while(1)
{
if((nbytes=recv(sockfd,buf,100,0))!=0)
{
buf[nbytes]='\0';
printf("RECEIVED FROM CLIENT\n");
printf("THE FILE CONTENTS ARE %s",buf);
fflush(stdout);
}
else
break;
}
close(sockfd);
return 0;
}
```

OUTPUT:

SERVER SIDE

```
[s@localhost ~]$ vi myfile
[s@localhost ~]$ cc nex8cser.c
[s@localhost ~]$ ./a.out
FILE TRANSFER
```

CLIENT SIDE

```
[s@localhost ~]$ cc nex8ccli.c
[s@localhost ~]$ ./a.out 127.0.0.1
FILE TRANSFER
INPUT
ENTER THE FILE NAME
ftpfile

OUTPUT
RECEIVED FROM CLIENT
THE FILE CONTENTS ARE Have a good day!
```

SERVER SIDE

```
[s@localhost ~]$ vi myfile
[s@localhost ~]$ cc nex8cser.c
[s@localhost ~]$ ./a.out
FILE TRANSFER
FILE IS SENT AND READ SUCCESSFULLY
```

RESULT:

EX NO 5 Applications using TCP and UDP sockets like DNS

AIM: To write a C program to implement Domain Name System using TCP/UDP Sockets.

ALGORITHM:
SERVER

```
Start the program.
Create the Socket for the Server.
Bind the Socket to the Port.
Listen for the incoming client connection.
Receive the IP address from the client to be resolved.
Get the domain name from the client.
Check the existence of the domain in the server.
If domain matches then send the corresponding address to the client.
Stop the program execution.
```

CLIENT

```
Start the program.
Create the Socket for the client.
Connect the Socket to the server.
Send the hostname to the server to be resolved.
If the server responds the print the address and terminates the process.
```

PROGRAM
SERVER

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
main()
{
int sd,sd2,nsd,clilen,sport,len,i;
char sendmsg[20],recvmsg[20];
char ipid[20][20]={"172.15.64.66","172.15.44.55","172.15.33.44","172.15.22.33"};
char hostid[20][20]={"www.yahoo.com","www.google.com","www.hotmail.com"};
struct sockaddr_in servaddr,cliaddr;
printf("DNS Server Side\n");
printf("Enter the Port\n");
scanf("%d",&sport);
sd=socket(AF_INET,SOCK_STREAM,0);
if(sd<0)
printf("Can't Create \n");
else
printf("Socket is Created\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(sport);
sd2=bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
if(sd2<0)
printf("Can't Bind\n");
else
printf("\n Binded\n");
listen(sd,5);
clilen=sizeof(cliaddr);
nsd=accept(sd,(struct sockaddr*)&cliaddr,&clilen);
if(nsd<0)
printf("Can't Accept\n");
else
printf("Accepted\n");
recv(nsd,recvmsg,20,0);
for(i=0;i<4;i++)
{
if(strcmp(recvmsg,hostid[i])==0)
{
send(nsd,ipid[i],20,20);
break;
}
}
}
```

CLIENT

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
main()
{
int csd,cport,len;
char sendmsg[20],recvmsg[20];
struct sockaddr_in servaddr;
printf("DNS Client Side\n");
printf("Enter the Client port\n");
scanf("%d",&cport);
csd=socket(AF_INET,SOCK_STREAM,0);
if(csd<0)
printf("Can't Create\n");
else
printf("Socket is Created\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(cport);
if(connect(csd,(struct sockaddr*)&servaddr,sizeof(servaddr))<0)
printf("Can't Connect\n");
else
printf("Connected\n");
printf("Enter the host address\n");
scanf("%s",sendmsg);
send(csd,sendmsg,20,0); recv(csd,recvmsg,20,20);
printf("The Corresponding IP Address is\n");
printf("%s",recvmsg);
}
```

OUTPUT

SERVER

```
[s@localhost ~]$ cc nex9aser.c
[s@localhost ~]$ ./a.out
DNS Server Side
Enter the Port
6770
Socket is Created

 Binded
Accepted
```

CLIENT

```
[s@localhost ~]$ cc nex9acli.c
[s@localhost ~]$ ./a.out
DNS Client Side
Enter the Client port
6770
Socket is Created
Connected
Enter the host address
www.google.com
The Corresponding IP Address is
172.15.33.44
```

RESULT:

EX NO 6 SIMULATING ARP / RARP PROTOCOLS

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

ALGORITHM:
SERVER

1. Include header files.
2. Create the socket address structure.
3. IP address, port number and family is initialized with server's socket address structure
4. Socket address is manipulated using the byte manipulation function bzero().
5. Socket created using socket() function.
6. Using bind () function, socket is bound with the server's well known port.
7. Enter number of client, accept those connection with those client with specified port.
8. The IP address is received from the client and the corresponding MAC address is sent to the client.
9. Socket is closed.

CLIENT

1. Include header files.
2. Socket address structure is created.
3. Server's socket address structure is initialized with IP address, port number and family.
4. Server and the client's address are manipulated using the byte manipulation functionbzero().
5. Socket is created using socket () function.
6. Socket is bound with the server's well known port.
7. The IP address is sent to the server.
8. Corresponding MAC address is received from the server and it is printed.
9. Socket is closed.

PROGRAM:

SERVER:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/shm.h>
#include<string.h>
main()
        {
                int shmid, a, i;
                char *ptr, *shmptr;
                shmid=shmget(3000,10,IPC_CREAT | 0666);
                shmptr=shmat(shmid,NULL,0);
                ptr=shmptr;
                for(i=0;i<3;i++)
                {
                        puts("Enter the Mac address");
                        scanf("%s",ptr);
```

```
                    a=strlen(ptr);
                    printf("string length:%d",a);
                    ptr[a]= ' ' ;
                    puts("Enter IP address");
                    ptr=ptr+a+1;
                    scanf("%s",ptr);
                    ptr[a]='\n' ;
                    ptr= ptr+a+1;
            }
            ptr[strlen(ptr)]= '\0';
            printf("\n ARP table at serverside is=\n%s", shmptr);
            shmdt(shmptr);
    }
```

CLIENT:

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/shm.h>
main()
    {
            int shmid,a;
            char *ptr, *shmptr;
            char ptr2[51], ip[12], mac[26];
            shmid=shmget(3000,10,0666);
            shmptr=shmat(shmid,NULL,0);
            puts("ARP table is");
            printf("%s",shmptr);
            printf("\n1.ARP\n 2.RARP\n 3.EXIT\n");
            scanf("%d",&a);
            switch(a)
            {
                    case 1:
                            puts("Enter IP address");
                            scanf("%s",ip);
                            ptr=strstr(shmptr, ip);
                            ptr-=8;
                            sscanf(ptr,"%s%*s",ptr2);
                            printf("MAC addr is %s",ptr2);
                            break;
                    case 2:
                            puts("Enter MAC addr");
                            scanf("%s",mac);
                            ptr=strstr(shmptr, mac);
                            sscanf(ptr,"%*s%s",ptr2);
                            printf("IP addr is %s",ptr2);
                            break;
                    case 3:
                            exit(1);  }
            }
```

OUTPUT:

SERVER SIDE

```
s@localhost ~]$ cc nex3a.c

[s@localhost ~]$ ./a.out

Enter the Mac address
a.a.a.a
String length:7
Enter IP address
1.1.1.1
Enter the Mac address
b.b.b.b
String length:7
Enter IP address
2.2.2.2
Enter the Mac address
c.c.c.c
String length:7
Enter IP address
3.3.3.3
 ARP table at serverside is=
a.a.a.a 1.1.1.1
b.b.b.b 2.2.2.2
c.c.c.c 3.3.3.3
```

CLIENT SIDE

```
[s@localhost ~]$ cc nex3cli.c
[s@localhost ~]$ ./a.out
ARP table is
a.a.a.a  1.1.1.1
b.b.b.b 2.2.2.2
c.c.c.c 3.3.3.3
1.ARP
2.RARP
3.EXIT
Enter your choice       1
Enter IP address
2.2.2.2
MAC addr is b.b.b.b
1.ARP
2.RARP
3.EXIT
Enter your choice       2
Enter MAC addr
c.c.c.c
IP addr is
3.3.3.3
1.ARP
2.RARP
3.EXIT
Enter your choice       3
```

RESULT:

EX NO 7A STUDY OF NETWORK SIMULATOR (NS2)

AIM: To study of ns2 in detail.

DESCRIPTION:

NS is an object oriented simulator, written in C++, with an OTcl interpreter as a frontend. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy in this document), and a similar class hierarchy within the OTcl interpreter (also called the interpreted hierarchy in this document). The two hierarchies are closely related to each other; from the user's perspective, there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy.

The root of this hierarchy is the class TclObject. Users create new simulator objects through the interpreter; these objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class hierarchy is automatically established through methods defined in the class TclClass. User instantiated objects are mirrored through methods defined in the class Tcl Object. There are other hierarchies in the C++ code and OTcl scripts; these other hierarchies are not mirrored in the manner of TclObject.

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 canefficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets.For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important. On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios.

In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important. ns meets both of these needs with two languages, C++ and OTcl. C++ is fast to run but slower to change, making it suitable for detailed protocol implementation. OTcl runs much slower but can be changed very quickly (and interactively), making it ideal for simulation configuration. ns (via tclcl) provides glue to make objects and variables appear on both languages.

SIMULATOR INITIALIZATION:

When a new simulation object is created in tcl, the initialization procedure performs the following operations:

• initialize the packet format (calls create_packetformat)

• create a scheduler (defaults to a calendar scheduler)

• create a "null agent" (a discard sink used in various places)

SCHEDULERS AND EVENTS:

The simulator is an event-driven simulator. There are presently four schedulers available in the simulator, each of which is implemented using a different data structure: a simple linked-list, heap, calendar queue (default), and a special type call called “real-time”. Each of these is described below. The scheduler runs by selecting the next earliest event, executing it to completion and returning to execute the next event. Unit of time used by scheduler is seconds. Presently, the simulator is single- threaded and only one event in execution at any given time. If more than one event are scheduled to execute at the same time, their execution is performed on the first scheduled – first dispatched manner. Simultaneous events are not reordered anymore by schedulers (as it was in earlier versions) and all schedulers should yeild the same order of dispatching given the same input.

NODE BASICS:
The basic primitive for creating a node is

```
set ns [new Simulator]

$ns node
```

The instance procedure node constructs a node out of simpler classifier objects (Section 5.4). The Node itself is a standalone class in OTcl. However, most of the components of the node are themselves TclObjects. The typical structure of a (unicast) node is as shown in Figure 5.1. This simple structure consists of two TclObjects: an address classifer (classifer_) and a port classifier (dmux_). The function of these classifiers is to distribute incoming packets to the correct agent or outgoing link.

All nodes contain at least the following components:

• an address or id_, monotonically increasing by 1 (from initial value 0) across the simulation namespace as nodes are created,

• a list of neighbors (neighbor_)

BASIC COMMANDS IN NS2:

- set *ns* [new Simulator]: generates an NS simulator object instance, and assigns it to variable *ns*.The "Simulator" object has member functions that do the following:
  - Create compound objects such as nodes and links (described later)
  - Connect network component objects created (ex. attach-agent)
  - Set network component parameters (mostly for compound objects)
  - Create connections between agents (ex. make connection between a "tcp" and "sink")
  - Specify NAM display options
  - Most of member functions are for simulation setup (referred to as plumbing functions in the Overview section) and scheduling, however some of them are for the NAM display. The "Simulator" object member function implementations are located in the file "ns-2/tcl/lib/ns-lib.tcl"

- *$ns* color *fid color*: is to set color of the packets for a flow specified by the flow id (fid). This member function of "Simulator" object is for the NAM display, and has no effect on the actual simulation.
- *$ns* namtrace-all *file-descriptor*: This member function tells the simulator to record simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command *$ns* flush-trace. Similarly, the member function trace-all is for recording the simulation trace in a general format

- proc*finish* {}: is called after this simulation is over by the command *$ns* at 5.0 "*finish*". In this function, post-simulation processes are specified.

- set*n0* [*$ns* node]: The member function node creates a node. A node in NS is compound object made of address and port classifiers (described in a later section). Users can create a node by separately creating an address and a port classifier objects and connecting them together. However, this member function of Simulator object makes the job easier.

- To see how a node is created, look at the files: "ns-2/tcl/libs/ns-lib.tcl"andns-2/tcl/libs/ns-node.tcl".
  $ns duplex-link *node1 node2 bandwidth delay queue-type*: creates two simplex links of specified bandwidth and delay, and connects the two specified nodes. In NS, the output queue of a node is implemented as a part of a link, therefore users should specify the queue-type when creating links. In the above simulation script, DropTail queue is used. If the reader wants to use a RED queue, simply replace the word DropTail with RED. The NS implementation of a link is shownin a later section. Like a node, a link is a compound object, and users can create its sub-objects and connect them and the nodes. Link source codes can be found in "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl" files. One thing to note is that you can insert error modules in a link component to simulate a lossy link (actually users can make and insert any network objects)

- *$ns* queue-limit *node1 node2 number*: This line sets the queue limit of the two simplex links that connect node1 and node2 to the number specified. At this point, the authors do not know how many of these kinds of member functions of Simulator objects are available and what they are. Please take a look at "ns-2/tcl/libs/ns-lib.tcl" and "ns-2/tcl/libs/ns-link.tcl", or NS.

- $ns duplex-link-op node1 node2 ...: The next couple of lines are used for the NAM display. To see the effects of these lines, users can comment these lines out and try the simulation.

  Now that the basic network setup is done, the next thing to do is to setup traffic agents such as TCP and UDP, traffic sources such as FTP and CBR, and attach them to nodes and agents respectively.

- set*tcp* [new *Agent/TCP*]: This line shows how to create a TCP agent. But in general, users can create any agent or traffic sources in this way. Agents and traffic sources are in fact basic objects (not compound objects), mostly implemented in C++ and linked to OTcl. Therefore, there are no specific Simulator object member functions that create these object instances. To create agents or traffic sources, a user should know the class names these objects (Agent/TCP, Agnet/TCPSink, Application/FTP and so on). This information can be found in the NS documentation or partly in this documentation. But one shortcut is to look at the "ns2/tcl/libs/ns-default.tcl" file. This file contains the default configurable parameter value settings for available network objects. Therefore,it works as a good indicator of what kind of network objects is available in NS and what are the configurable parameters.

- *$ns* attach-agent *node agent*: The attach-agent member function attaches an agent object created to a node object. Actually, what this function does is call the attach member function of specified node, which attaches the given agent to itself. Therefore, a user can do the same thing by, for example, $n0 attach $tcp. Similarly, each agent object has a member function .

- *$ns* connect *agent1 agent2*: After two agents that will communicate with each other are created, the next thing is to establish a logical network connection between them. This line establishes a network connection by setting the destination address to each others' network and port address pair.Assuming that all the network configuration is done, the next thing to do is write a simulation scenario (i.e. simulation scheduling). The Simulator object has many scheduling member functions. However, the one that is mostly used is the following:

- *$ns* at *time "string"*: This member function of a Simulator object makes the scheduler (scheduler_ is the variable that points the scheduler object created by [new Scheduler] command at the beginning of the script) to schedule the execution of the specified string at given simulation time. For example, *$ns* at *0.1 "$cbr start"* will make the scheduler call a start member function of the CBR traffic source object, which starts the CBR to transmit data. In NS, usually a traffic source does not transmit actual data, but it notifies the underlying agent that it has some amount of data to transmit, and the agent, just knowing how much of the data to transfer, creates packets and sends them.

  After all network configurations, scheduling and post-simulation procedure specifications are done, the only thing left is to run the simulation. This is done by *$ns* run.

RESULT:

EX NO 7B STUDY OF TCP/UDP PERFORMANCE USING SIMULATION TOOL

AIM:

To study the performance comparison of User Datagram Protocol and Transmission Control Protocol

UDP

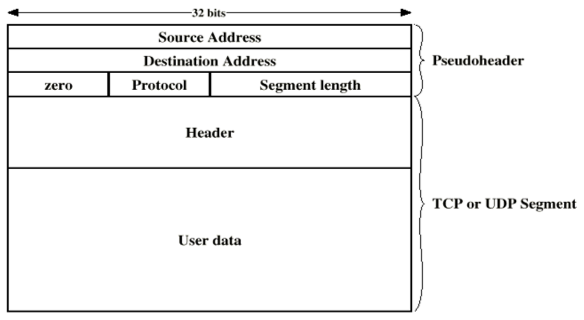
Introduction:

The User Datagram Protocol (UDP) is one of the core members of the Internet Protocol Suite, the set of network protocols used for the Internet. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths. UDP is sometimes called the Universal Datagram Protocol. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP uses a simple transmission model without implicit hand-shaking dialogues for guaranteeing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers).Common network applications that use UDP include: the Domain Name System (DNS), streaming media applications such as IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and many online games.

Packet structure :



UDP is a minimal message-oriented Transport Layer protocol that is documented in IETF RFC 768.UDP provides no guarantees to the upper layer protocol for message delivery and the UDP protocol layer retains no state of UDP messages once sent. For this reason, UDP is sometimes referred to as Unreliable Datagram Protocol. UDP provides application multiplexing (via port numbers) and integrity verification (via checksum) of the header and payload. If transmission reliability is desired, it must be implemented in the user's application.

The UDP header consists of 4 fields, all of which are 2 bytes (16 bits).

Reliability and congestion control solutions:

Lacking reliability, UDP applications must generally be willing to accept some loss, errors or duplication. Some applications such as TFTP may add rudimentary reliability mechanisms into the application layer as needed.[2] Most often, UDP applications do not require reliability mechanisms and may even be hindered by them. Streaming media, real-time multiplayer games and voice over IP (VoIP) are examples of applications that often use UDP. If an application requires a high degree of reliability, a protocol such as the Transmission Control Protocol or erasure codes may be used instead.

Lacking any congestion avoidance and control mechanisms, network-based mechanisms are required to minimize potential congestion collapse effects of uncontrolled, high rate UDP traffic loads. In other words, since UDP senders cannot detect congestion, network-based elements such as routers using packet queuing and dropping techniques will often be the only tool available to slow down excessive UDP traffic. The Datagram Congestion Control Protocol (DCCP) is being designed as a partial solution to this potential problem by adding end host TCP-friendly congestion control behavior to high-rate UDP streams such as streaming media.

Comparison of UDP and TCP:

Transmission Control Protocol is a connection-oriented protocol, which means that it requires handshaking to set up end-to-end communications. Once a connection is set up user data may be sent bi-directionally over the connection.

Reliable – TCP manages message acknowledgment, retransmission and timeout. Multiple attempts to deliver the message are made. If it gets lost along the way, the server will re-request the lost part. In TCP, there's either no missing data, or, in case of multiple timeouts, the connection is dropped.

Ordered – if two messages are sent over a connection in sequence, the first message will reach the receiving application first. When data segments arrive in the wrong order, TCP buffers the out-of-order data until all data can be properly re-ordered and delivered to the application.

Heavyweight – TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.

Streaming – Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.

UDP is a simpler message-based connectionless protocol. Connectionless protocols do not set up a dedicated end-to-end connection. Communication is achieved by transmitting information in one direction from source to destination without verifying the readiness or state of the receiver.

Unreliable – When a message is sent, it cannot be known if it will reach its destination; it could get lost along the way. There is no concept of acknowledgment, retransmission or timeout.

Not ordered – If two messages are sent to the same recipient, the order in which they arrive cannot be predicted.

Lightweight – There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.

Datagrams – Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent.

TCP

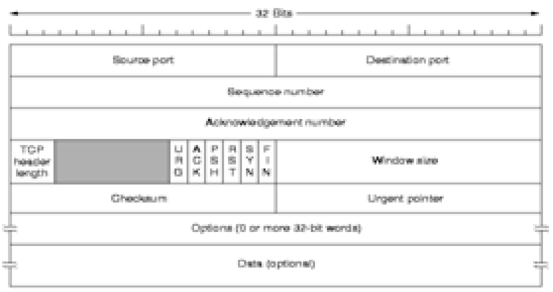
Introduction :

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite (the other being Internet Protocol, or IP), so the entire suite is commonly referred to as TCP/IP.

TCP segment structure:

A TCP segment consists of a segment header and a data section. The TCP header contains 10 mandatory fields, and an optional extension field (Options, pink background in table).

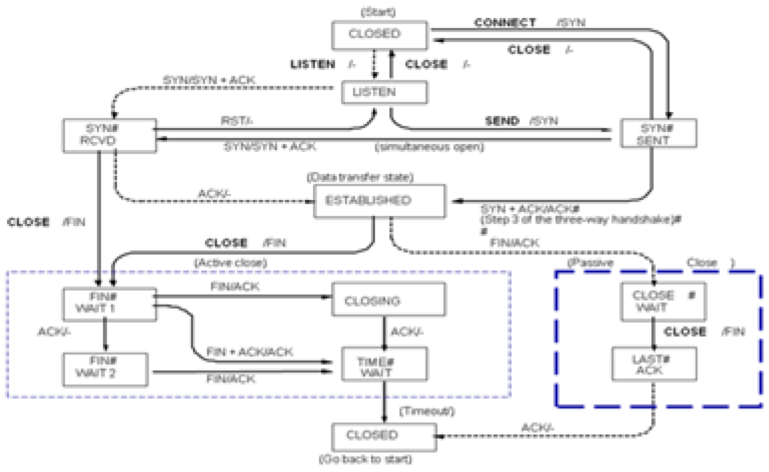
The data section follows the header. Its contents are the payload data carried for the application. The length of the data section is not specified in the TCP segment header. It can be calculated by subtracting the combined length of the TCP header and the encapsulating IP segment header from the total IP segment length (specified in the IP segment header).



Protocol operation:

A Simplified TCP State Diagram shown below. See TCP EFSM diagram for a more detailed state diagram including the states inside the ESTABLISHED state. TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (connection establishment) before entering the data transfer phase. After data transmission is completed, the connection termination closes established virtual circuits and releases all allocated resources.

During the lifetime of a TCP connection it undergoes a series of state changes:



LISTEN : In case of a server, waiting for a connection request from any remote client.

SYN-SENT : waiting for the remote peer to send back a TCP segment with the SYN and ACK flags set. (usually set by TCP clients)

SYN-RECEIVED : waiting for the remote peer to send back an acknowledgment after having sent back a connection acknowledgment to the remote peer. (Usually set by TCP servers)

ESTABLISHED : the port is ready to receive/send data from/to the remote peer.

FIN-WAIT-1

FIN-WAIT-2

CLOSE-WAIT

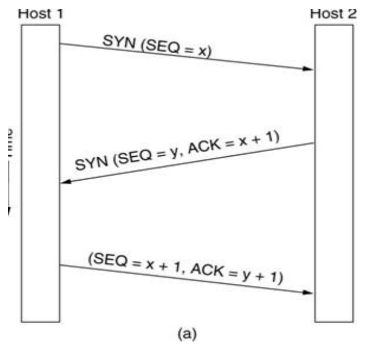
CLOSING

LAST-ACK

TIME-WAIT: represents waiting for enough time to pass to be sure the remote peer received the acknowledgment of its connection termination request. According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes.

CLOSED

Connection establishment:



(a)

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

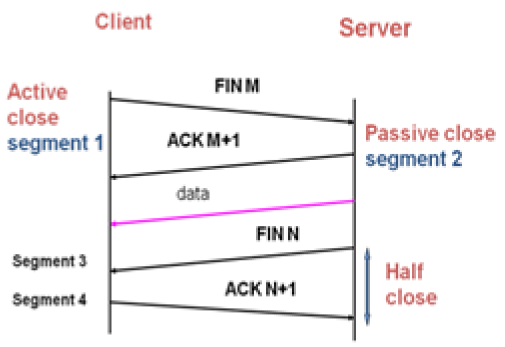
The active open is performed by the client sending a SYN to the server. It sets the segment's sequence number to a random value A.

In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number (A + 1), and the sequence number that the server chooses for the packet is another random number, B.

Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. A + 1, and the acknowledgement number is set to one more than the received sequence number i.e. B + 1.

At this point, both the client and server have received an acknowledgment of the connection.

Connection Termination:



The connection termination phase uses, at most, a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. Therefore, a typical tear-down requires a pair of FIN and ACK segments from each TCP endpoint.

A connection can be "half-open", in which case one side has terminated its end, but the other has not. The side that has terminated can no longer send any data into or receive any data from the connection, but the other side can (but generally if it tries, this should result in no acknowledgment and therefore a timeout, or else result in a positive RST and either way thereby the destruction of the half- open socket).

It is also possible to terminate the connection by a 3-way handshake, when host A sends a FIN and host B replies with a FIN & ACK (merely combines 2 steps into one) and host A replies with an ACK.[13] This is perhaps the most common method.

Result:

EX NO 8A IMPLEMENTATION OF DISTANCE VECTOR ROUTING PROTOCOL

AIM: To perform the simulation of the distance vector routing protocol using NS2.

ALGORITHM:

1. Start the program
2. Create the trace file and NAM file
3. Setup the topology object
4. Create mobile nodes and attach them to the channel
5. Configure the nodes and provide initial location of mobile nodes
6. Set up a TCP Connection between nodes
7. Define and initialize positions for the NAM window
8. Specify the end of simulation
9. Stop.
10. 10.

PROGRAM:

```
set ns [new Simulator]

#Define different colors for data flows (for NAM)

$ns color 1 Blue

$ns color 2 Yellow

#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 0

}
```

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

```
$ns rtproto DV

#Create six nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]
```

#Create links between the nodes

```
$ns duplex-link $n0 $n4 0.3Mb 10ms DropTail

$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 $n4 $n3 0.5Mb 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 $n4 orient up

$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 $n4 $n3 orient right

$ns duplex-link-op $n3 $n5 orient left-up

$ns duplex-link-op $n4 $n5 orient right-up
```

#Setup a TCP connection

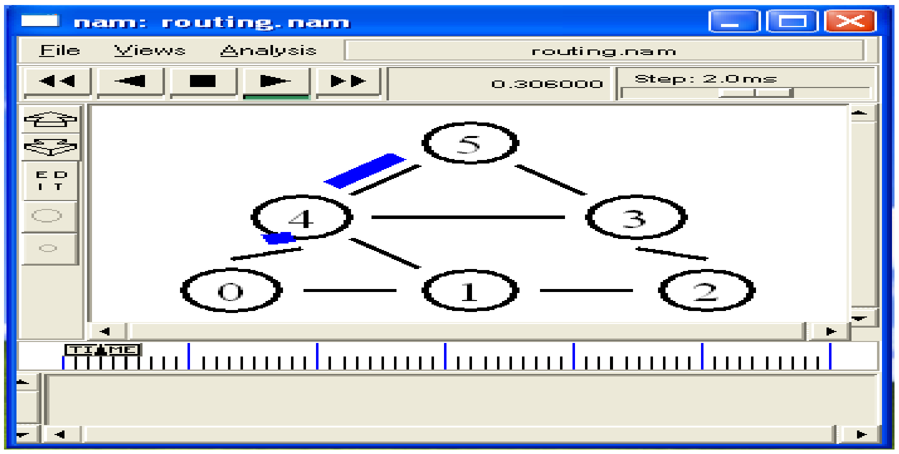
```
set tcp [new Agent/TCP/Newreno]

$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink/DelAck]

$ns attach-agent $n5 $sink

$ns connect $tcp $sink

$tcp set fid_ 1
```

#Setup a FTP over TCP connection

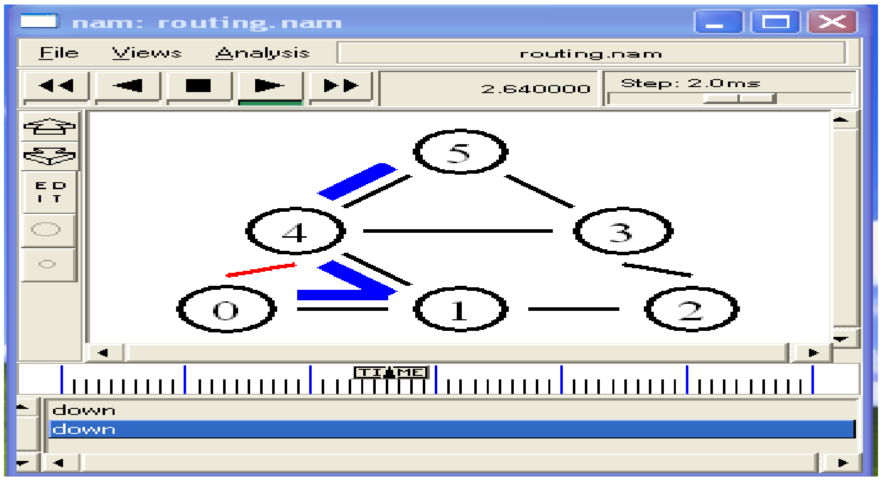
```
set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ftp set type_ FTP

$ns rtmodel-at 1.0 down $n0 $n4

$ns rtmodel-at 4.5 up $n0 $n4

$ns at 0.1 "$ftp start"

$ns at 6.0 "finish"

$ns run
```

OUTPUT:

Up:

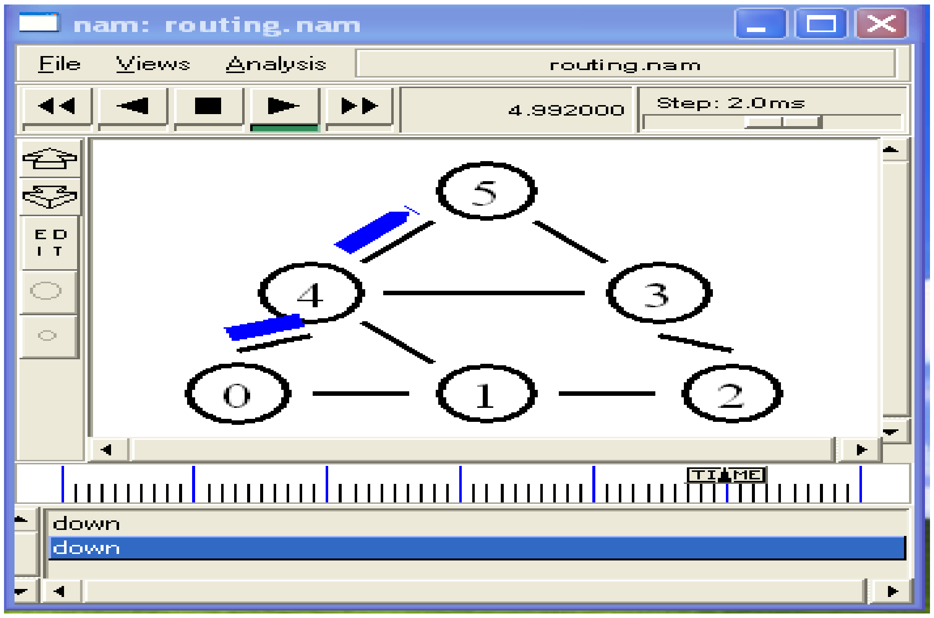


Down:



```
down
down
```

Up:



```
down
down
```

RESULT:

EX: NO: 8B IMPLEMENTATION OF LINK STATE ROUTING PROTOCOL

AIM: To perform the simulation of the link state routing protocol using NS2.

ALGORITHM:

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

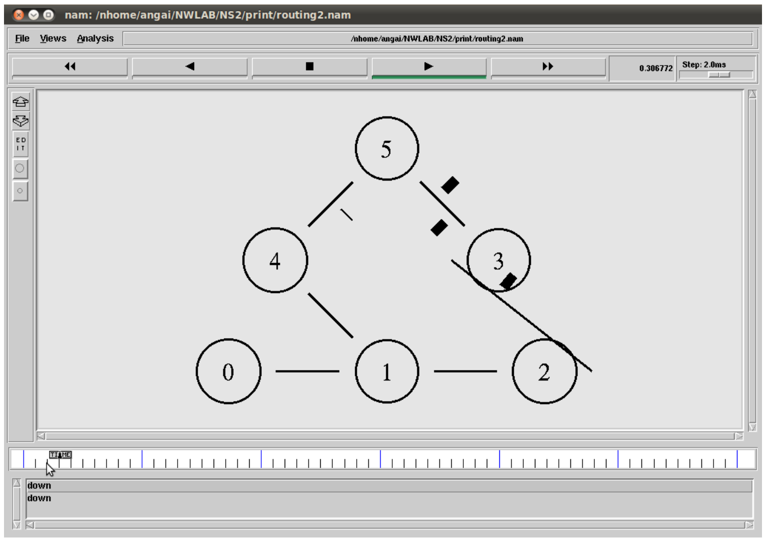
PROGRAM:

```
#routing2.tcl

set ns [new Simulator]


#Define different colors for data flows (for NAM)

$ns color 1 Blue

$ns color 2 Red


#Open the Trace file

set file1 [open routing2.tr w]

$ns trace-all $file1


#Open the NAM trace file

set file2 [open routing2.nam w]

$ns namtrace-all $file2
```

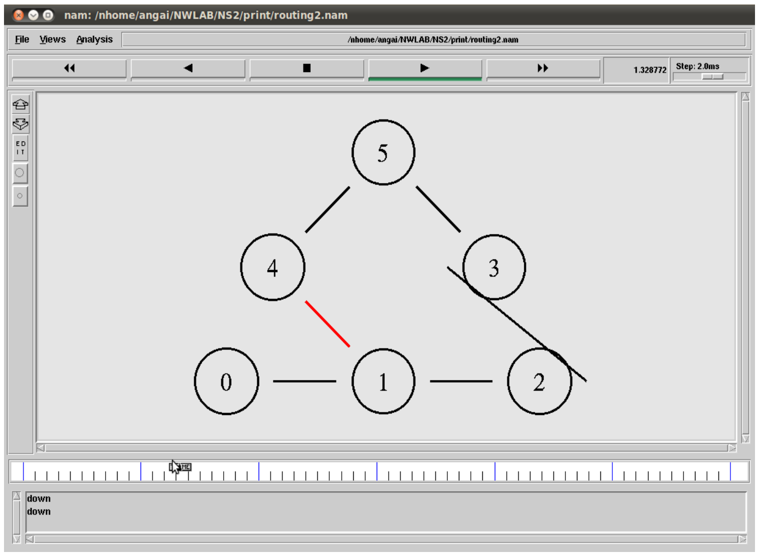
```
#Define a 'finish' procedure

proc finish {}

{

global ns file1 file2

        $ns flush-trace

close $file1

close $file2

exec nam routing2.nam &

exit 0

}

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

$ns rtproto LS

#Create six nodes

set n0 [$ns node]

set n1 [$ns node]

set n2 [$ns node]

set n3 [$ns node]

set n4 [$ns node]

set n5 [$ns node]


#Create links between the nodes

$ns duplex-link $n0 $n1 0.3Mb 10ms DropTail

$ns duplex-link $n1 $n2 0.3Mb 10ms DropTail

$ns duplex-link $n2 $n3 0.3Mb 10ms DropTail

$ns duplex-link $n1 $n4 0.3Mb 10ms DropTail


$ns duplex-link $n3 $n5 0.5Mb 10ms DropTail

$ns duplex-link $n4 $n5 0.5Mb 10ms DropTail
```

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

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

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

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


$ns at 0.1 "$ftp start"
$ns at 6.0 "finish"
$ns run
```

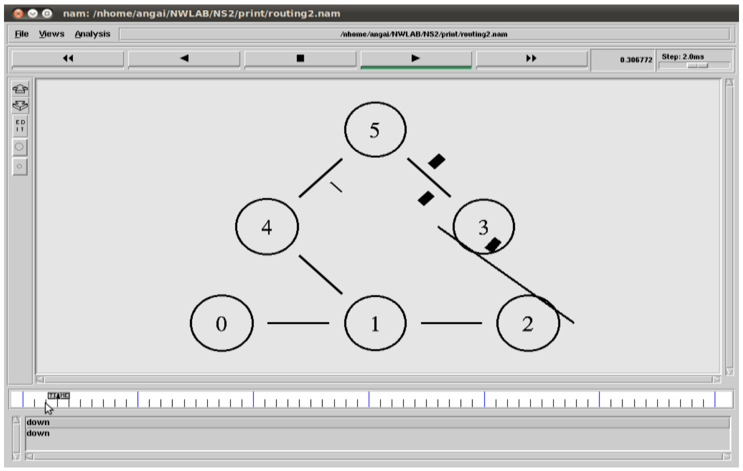
OUTPUT:


nam: /nhome/angai/NWLAB/NS2/print/routing2.nam
File Views Analysis
/nhome/angai/NWLAB/NS2/print/routing2.nam
0.306772
Step: 2.0ms
5
4
3
0
1
2
down
down

Before Link failure between Nodes n1 and n4


nam: /nhome/angai/NWLAB/NS2/print/routing2.nam
File Views Analysis
/nhome/angai/NWLAB/NS2/print/routing2.nam
1.328772
Step: 2.0ms
5
4
3
0
1
2
down
down

While Link failure between Nodes n1 and n4


nam: /nhome/angai/NWLAB/NS2/print/routing2.nam
File Views Analysis
/nhome/angai/NWLAB/NS2/print/routing2.nam
0.306772
Step: 2.0ms
5
4
3
0
1
2
down
down

After failed link between Nodes n1 and n4 up

RESULT:

EX NO 9 PERFORMANCE EVALUATION OF ROUTING PROTOCOLS USING SIMULATION TOOL

AIM:

To perform the simulation of the routing protocols using NS2.

ALGORITHM:

Step 1: Start the program

Step 2: Create the trace file and NAM file

Step 3: Setup the topology object

Step 4: Create mobile nodes and attach them to the channel

Step 5: Configure the nodes and provide initial location of mobile nodes

Step 6: Set up a TCP Connection between nodes

Step 7: Define and initialize positions for the NAM window

Step 8: Specify the end of simulation

Step 9: Stop.

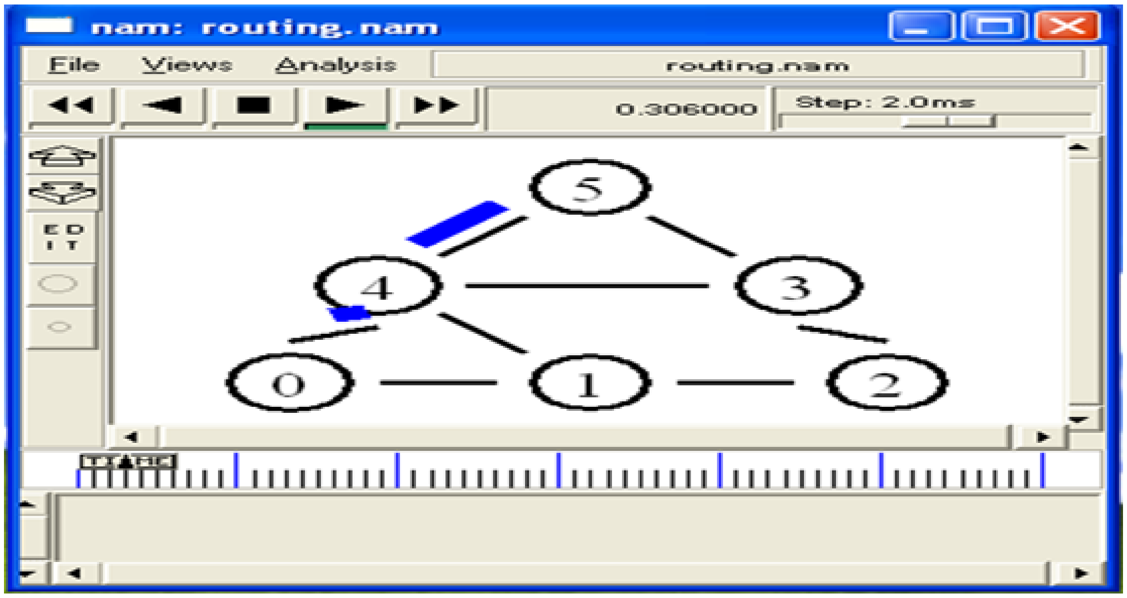
PROGRAM

```
set ns [new Simulator]
#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Yellow
#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 0
}

# Next line should be commented out to have the static routing
$ns rtproto DV
#Create six nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n4 0.3Mb 10ms DropTail
$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 $n4 $n3 0.5Mb 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 $n4 orient up
$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 $n4 $n3 orient right
$ns duplex-link-op $n3 $n5 orient left-up
$ns duplex-link-op $n4 $n5 orient right-up
```

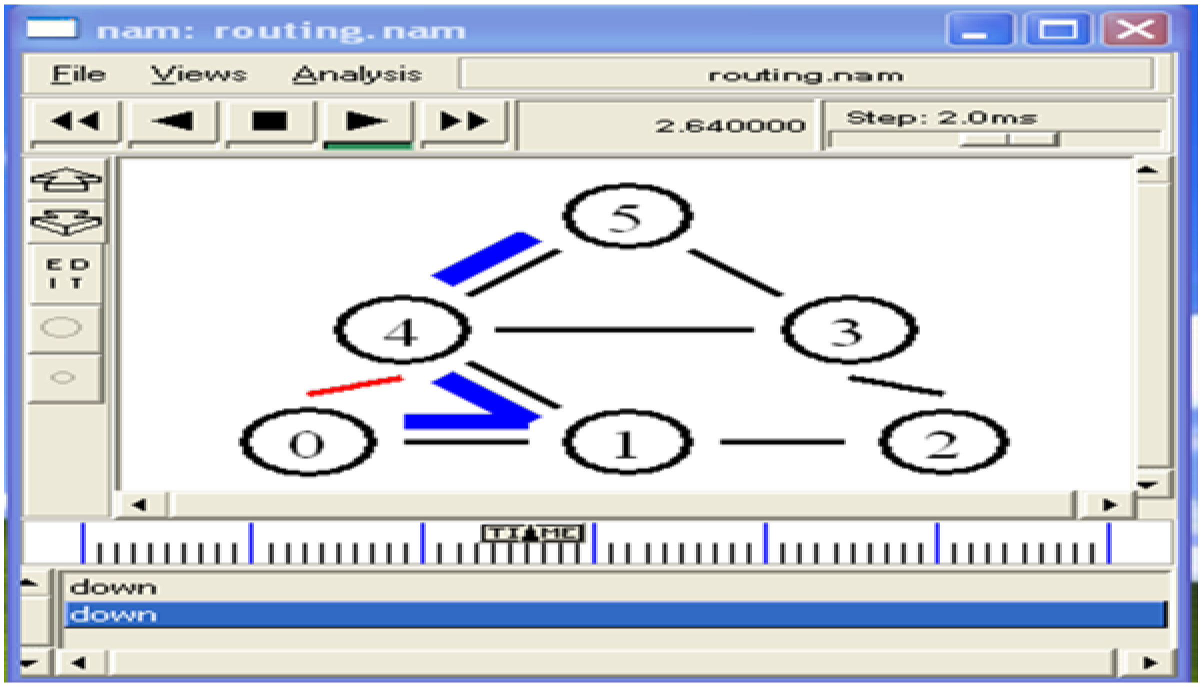
```
#Setup a TCP connection
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n5 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP
$ns rtmodel-at 1.0 down $n0 $n4
$ns rtmodel-at 4.5 up $n0 $n4
$ns at 0.1 "$ftp start"
$ns at 6.0 "finish"
$ns run
```

Output:

UP:

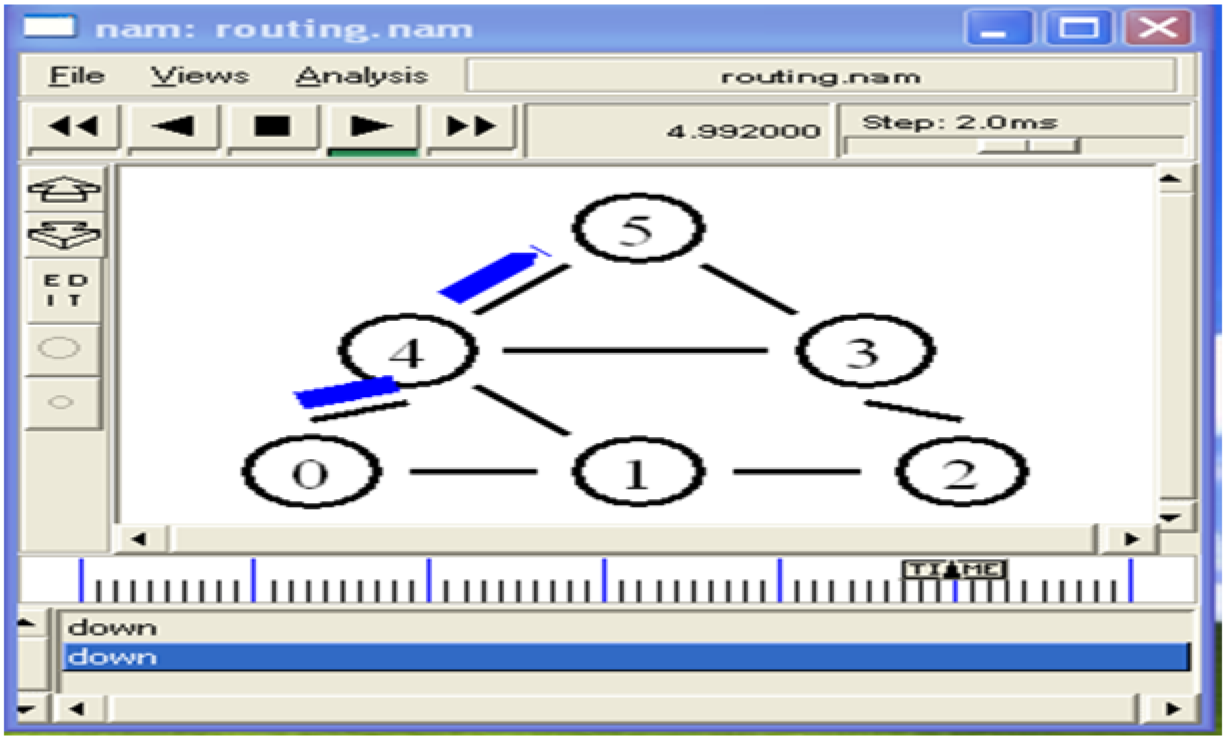


Down:



down
down

UP:



down
down

Result:.

EX NO 10(i) SIMULATIONOFFLOODING

AIM:

To simulate the performance of flooding using ns2.

ALGORITHM:

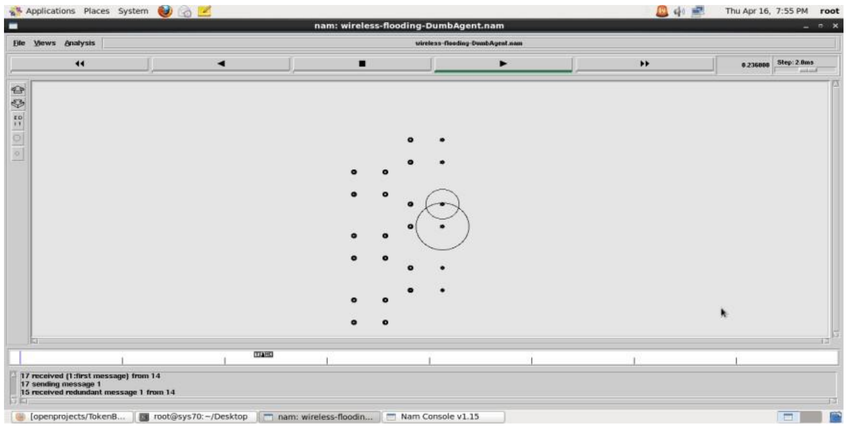
1. Start the program
2. Create the trace file and NAM file.
3. Setup the flooding agent
4. Create nodes and set duplex links between the nodes.
5. Create nodes and attach them to the queue, DROPTAIL
6. Configure the nodes and provides initial location of nodes. Generation of movements is done.
7. Setup connection between the nodes
8. Define a initialize positions for the nam window.
9. Telling nodes when the simulation ends.
10. Ending nam and the simulation.

PROGRAM:

```
Mac/Simple set bandwidth_ 1Mb
set MESSAGE_PORT 42
set BROADCAST_ADDR -1
# variables which control the number of nodes and how they're grouped
# (see topology creation code below)
set group_size 4
set num_groups 6
set num_nodes [expr $group_size * $num_groups]
set val(chan) Channel/WirelessChannel ;#Channel Type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(netif) Phy/WirelessPhy ;# network interface type
#set val(mac) Mac/802_11 ;# MAC type
#set val(mac) Mac ;# MAC type
set val(mac) Mac/Simple
set val(ifq) Queue/DropTail/PriQueue ;# interface queue type
set val(ll) LL ;# link layer type
set val(ant) Antenna/OmniAntenna ;# antenna model
set val(ifqlen) 50 ;# max packet in ifq
# DumbAgent, AODV, and DSDV work. DSR is broken
```

```
set val(rp) DumbAgent
#set val(rp) DSDV
#set val(rp) DSR
#set val(rp) AODV
# size of the topography
set val(x) [expr 120*$group_size + 500]
set val(y) [expr 240*$num_groups + 200]
set ns [new Simulator]
set f [open wireless-flooding-$val(rp).tr w]
$ns trace-all $f
set nf [open wireless-flooding-$val(rp).nam w]
$ns namtrace-all-wireless $nf $val(x) $val(y)
$ns use-newtrace
# set up topography object
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
#
# Create God
#
create-god $num_nodes
set chan_1_ [new $val(chan)]
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace OFF \
-macTrace ON \
-movementTrace OFF \
-channel $chan_1_
# subclass Agent/MessagePassing to make it do flooding
Class Agent/MessagePassing/Flooding -superclass Agent/MessagePassing
Agent/MessagePassing/Flooding instproc recv {source sport size data} {
$self instvar messages_seen node_
global ns BROADCAST_ADDR
# extract message ID from message
set message_id [lindex [split $data ":"] 0]
puts "\nNode [$node_ node-addr] got message $message_id\n"
if {[lsearch $messages_seen $message_id] == -1} {
lappend messages_seen $message_id
$ns trace-annotate "[$node_ node-addr] received {$data} from $source"
$ns trace-annotate "[$node_ node-addr] sending message $message_id"
$self sendto $size $data $BROADCAST_ADDR $sport
} else {
$ns trace-annotate "[$node_ node-addr] received redundant message $message_id from $source"
}
}
Agent/MessagePassing/Flooding instproc send_message {size message_id data port} {
$self instvar messages_seen node_
global ns MESSAGE_PORT BROADCAST_ADDR
lappend messages_seen $message_id
$ns trace-annotate "[$node_ node-addr] sending message $message_id"
$self sendto $size "$message_id:$data" $BROADCAST_ADDR $port
}
# create a bunch of nodes
for {set i 0} {$i < $num_nodes} {incr i} {
set n($i) [$ns node]
$n($i) set Y_ [expr 230*floor($i/$group_size) + 160*(($i%$group_size)>=($group_size/2))]
$n($i) set X_ [expr (90*$group_size)*($i/$group_size%2) + 200*($i%($group_size/2))]
$n($i) set Z_ 0.0
$ns initial_node_pos $n($i) 20
}
# attach a new Agent/MessagePassing/Flooding to each node on port $MESSAGE_PORT
for {set i 0} {$i < $num_nodes} {incr i} {
set a($i) [new Agent/MessagePassing/Flooding]
$n($i) attach $a($i) $MESSAGE_PORT
$a($i) set messages_seen {}
}
# now set up some events
$ns at 0.2 "$a(1) send_message 200 1 {first message} $MESSAGE_PORT"
$ns at 0.4 "$a([expr $num_nodes/2]) send_message 600 2 {some big message} $MESSAGE_PORT"
$ns at 0.7 "$a([expr $num_nodes-2]) send_message 200 3 {another one} $MESSAGE_PORT"
$ns at 1.0 "finish"
proc finish {} {
global ns f nf val
$ns flush-trace
close $f
close $nf
# puts "running nam..."
exec nam wireless-flooding-$val(rp).nam &
exit 0
}
$ns run
```

OUTPUT:



```
17 received (1:first message) from 14
17 sending message 1
15 received redundant message 1 from 14
```

RESULT:

EX: NO: 10(ii) SIMULATIONOFCONGESTIONCONTROL

AIM: To simulate the performance Stop wait protocol under congestion using NS2.

ALGORITHM:

1. Start the program
2. Create the trace file and NAM file.
3. Setup the topology object
4. Create nodes and set duplex links between the nodes.
5. Create nodes and attach them to the queue, DROPTAIL
6. Configure the nodes and provide initial location of nodes. Generation of movements is done.
7. Setup a TCP Connection between nodes
8. Define a initialize positions for the nam window.
9. Telling nodes when the simulation ends
10. Ending nam and the simulation

PROGRAM:

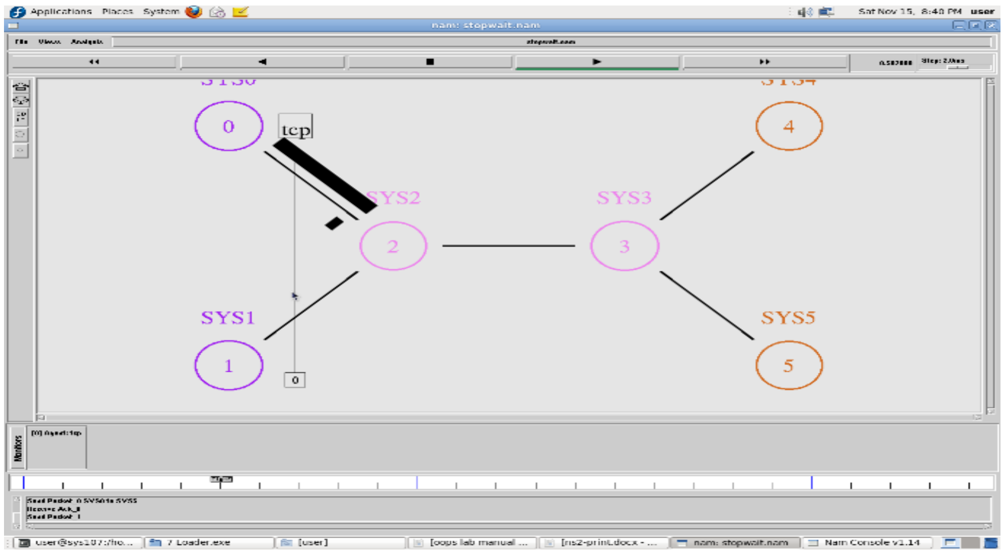
```
setns[newSimulator]

setn0[$nsnode]

setn1[$nsnode]

setn2[$nsnode]

setn3[$nsnode]

setn4[$nsnode]

setn5[$nsnode]

$n0color"purple"

$n1color"purple"

$n2color"violet"

$n3color"violet"

$n4color"chocolate"

$n5color"chocolate"

setf[openstopwait.trw]
```

```
$nstrace-all$f

setnf[openstopwait.namw]

$nsnamtrace-all$nf

$nsat0.0"$n0labelSYS0"

$nsat0.0"$n1labelSYS1"

$nsat0.0"$n2labelSYS2"

$nsat0.0"$n3labelSYS3"

$nsat0.0"$n4labelSYS4"

$nsat0.0"$n5labelSYS5"

$nsduplex-link$n0$n20.2Mb20msDropTail

$nsduplex-link$n1$n20.2Mb20msDropTail

$nsduplex-link$n2$n30.2Mb20msDropTail

$nsduplex-link$n3$n40.2Mb20msDropTail

$nsduplex-link$n3$n50.2Mb20msDropTail

$nsduplex-link-op$n0$n2orientright-down

$nsduplex-link-op$n1$n2orientright-up

$nsduplex-link-op$n2$n3orientright

$nsduplex-link-op$n3$n4orientright-up

$nsduplex-link-op$n3$n5orientright-down

$nsqueue-limit$n0$n210

Agent/TCPset_nam_tracevar_true

settcp[newAgent/TCP]

$tcpsetwindow1

$tcpsetmaxcwnd1

$tcpsetfid1

$nsattach-agent$n0$tcp
```

```
setsink[newAgent/TCPSink]

$nsattach-agent$n5$sink

$nsconnect$tcp$sink

setftp[newApplication/FTP]

$ftpattach-agent$tcp

$nsadd-agent-trace$tcptcp

$nsmonitor-agent-trace$tcp

$tcptracevarcwnd

$nsat0.1"$ftpstart"

$nsat0.53"$nsqueue-limit$n3$n50"

$nsat0.80"$nsqueue-limit$n3$n55"

$nsat2.0"$nsdetach-agent$n0$tcp;$nsdetach-agent$n5$sink"

$nsat2.5"finish"

#$nsat0.0"$nstrace-annotate\"STOPWAIT\""

$nsat0.01"$nstrace-annotate\"FTPstartsat0.01\""

$nsat0.10"$nstrace-annotate\"SendRequestSYS0toSYS5\""

$nsat0.18"$nstrace-annotate\"ReceiveRequestSYS5toSYS0\""

$nsat0.24"$nstrace-annotate\"SendPacket_0SYS0toSYS5\""

$nsat0.42"$nstrace-annotate\"ReceiveAck_0\""

$nsat0.48"$nstrace-annotate\"SendPacket_1\""

$nsat0.60"$nstrace-annotate\"DisconnectN2Solossthepacket1\""

$nsat0.67"$nstrace-annotate\"WaitingforAck_1\""

$nsat0.95"$nstrace-annotate\"SendPacket_1again\""

$nsat1.95"$nstrace-annotate\"DeattachSYS3,Packet_1again\""

$nsat2.09"$nstrace-annotate\"ReceiveAck_1\""

$nsat2.10"$nstrace-annotate\"SEndPacket_2\""
```

```
$nsat2.38"$nstrace-annotate\"ReceiveAck_2\""

$nsat2.5"$nstrace-annotate\"FTPstops\""

procfinish{}

{

globalnsnf

$nsflush-trace

close$nf

execnamstopwait.nam&

exit0

}

$nsrun
```

OUTPUT:



```
Send Packet_0 SYS0 to SYS5
Receive Ack_0
Send Packet_1
```

RESULT:

Ex.No:11 ERROR DETECTION TECHNIQUE- CYCLIC REDUNDANCY CHECK (CRC)

AIM: To implement the error detection technique cyclic redundancy check.

ALGORITHM:

1. Start the program
2. Enter the data and generate the polynomial.
3. Call the CRC function with performs XOR between the divisors and generated.
4. Display the checksum.
5. Once the final code is generated, error detection test can be made.
6. Enter the position where error is to be inserted and display the erroneous data.
7. Call the CRC and to check the error.
8. If the error exists, display “error detected” else “no error detected”.
9. Stop the program.

PROGRAM:

```
#include<stdio.h>

#include<string.h>

#define N strlen(g)



char t[28],cs[28],g[]="10001000000100001";

int a,e,c;



void xor()

{

for(c = 1;c < N; c++)

cs[c] = (( cs[c] == g[c])?'0':'1');

}



void crc()

{

for(e=0;e<N;e++)
```

```
cs[e]=t[e];
do
    {
if(cs[0]=='1')
xor();
for(c=0;c<N-1;c++)
cs[c]=cs[c+1];
cs[c]=t[e++];
}while(e<=a+N-1);
}

int main()
{
printf("\nEnter data : ");
scanf("%s",t);
printf("\n----------------------------------------");
printf("\nGeneratng polynomial : %s",g);
  a=strlen(t);
for(e=a;e<a+N-1;e++)
t[e]='0';
printf("\n----------------------------------------");
printf("\nModified data is : %s",t);
printf("\n----------------------------------------");
crc();
printf("\nChecksum is : %s",cs);
for(e=a;e<a+N-1;e++)
```

```
t[e]=cs[e-a];
printf("\n----------------------------------------");
printf("\nFinal codeword is : %s",t);
printf("\n----------------------------------------");
printf("\nTest error detection 0(yes) 1(no)? : ");
scanf("%d",&e);
if(e==0)
   {
do
         {
printf("\nEnter the position where error is to be inserted : ");
scanf("%d",&e);
}while(e==0 || e>a+N-1);
t[e-1]=(t[e-1]=='0')?'1':'0';
printf("\n----------------------------------------");
printf("\nErroneous data : %s\n",t);
   }
crc();
for(e=0;(e<N-1) && (cs[e]!='1');e++);
if(e<N-1)
printf("\nError detected\n\n");
else
printf("\nNo error detected\n\n");
printf("\n----------------------------------------\n");
return 0;
}
```

OUTPUT:

```
Enter data : 1101

--------------------------------------------

Generating polynomial : 10001000000100001

--------------------------------------------

Modified data is : 11010000000000000000

--------------------------------------------

Checksum is : 1101000110101101

--------------------------------------------

Final codeword is : 11011101000110101101

--------------------------------------------

Test error detection 0(yes) 1(no)? : 0

Enter the position where error is to be inserted : 3

--------------------------------------------

Erroneous data : 11111101000110101101

Error detected

-----------------------------------------
```

RESULT:

EX NO 12 APPLICATIONS USING TCP AND UDP SOCKETS LIKE SNMP

AIM: To write a C program for simulation of Simple Network management Protocols.

ALGORITHM:

MANAGER

1. Start the program.
2. Create an unnamed socket for client using socket ( ) system.
3. Call with parameters AF_INET as domain and SOCK_STREAM as type. Step 4: Name the socket using bind ( ) system call.
4. Now connect the socket to agent using connect ( ) system call.
5. Get the input for the type of information needed from the agent.
6. If the input is equal to „TCP connection" then goto next Step else If it is equal to „system" Goto Step 9.
7. Read the input for the object, send it and receive the details of the TCP connection of that object from the agent. Go to Step 10.
8. Read the input for the object, send it and receive the details of the system from the agent. Go to Step 10.
9. Receive the message, print and terminate the process.
10. Stop the process.

AGENTS

1. Start the program.
2. Create an unnamed socket for the server using the parameters AF_INET as domain and the SOCK_STREAM as type.
3. Name the socket using bind( ) system call with the parameters server_sockfd and the manager address(sin_addr and sin_sport).
4. Create a connection queue and wait for manager using the listen ( ) system call with the number of manager request as parameters.
5. Accept the connection using accept( ) system call when manager requests for connection.
6. Receive the message from the manager. If the request is for „TCP connections" then send the details of the requested object, else if the request is for „System" then send the details of the requested system.
7. Stop the program execution.

PROGRAM:
AGENT1

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
main()
{
```

```
int i,sd,sd2,nsd,clilen,sport,len;
char sendmsg[20],recvmsg[100];
char oid[5][10]={"client1","client2","client3","cleint4","client5"};
char wsize[5][5]={"5","10","15","3","6"};
struct sockaddr_in servaddr,cliaddr;
printf("I'm the Agent - TCP Connection\n");
printf("\nEnter the Server port");
printf("\n_____________________\n");
scanf("%d",&sport);
sd=socket(AF_INET,SOCK_STREAM,0);
if(sd<0)
printf("Can't Create \n");
else
printf("Socket is Created\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(sport);
sd2=bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
if(sd2<0)
printf(" Can't Bind\n");
else
printf("\n Binded\n");
listen(sd,5);
clilen=sizeof(cliaddr);
nsd=accept(sd,(struct sockaddr*)&cliaddr,&clilen);
if(nsd<0)
printf("Can't Accept\n");
else
printf("Accepted\n");
recv(nsd,recvmsg,100,0);
for (i=0;i<5;i++)
{
if(strcmp(recvmsg,oid[i])==0)
{
send(nsd,wsize[i],100,0); break;
}
}
}
```

AGENT2

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<string.h>
main()
{
int i,sd,sd2,nsd,clilen,sport,len;
char sendmsg[20],recvmsg[100];
char oid[5][10]={"System1","System2","System3","System4","System5"};
char mdate[5][15]={"1-10-05","10-03-11","14.03.14","11.07.13","17.12.10"};
char time[5][15]={"9am","10pm","11am","12.30pm","11.30am"};
struct sockaddr_in servaddr,cliaddr;
printf("Enter the Server port");
printf("\n____________________\n");
scanf("%d",&sport);
sd=socket(AF_INET,SOCK_STREAM,0);

if(sd<0)
printf("Can't Create \n");
else
printf("Socket is Created\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(sport);
sd2=bind(sd,(struct sockaddr*)&servaddr,sizeof(servaddr));
if(sd2<0)
printf(" Can't Bind\n");
else
printf("\n Binded\n");
listen(sd,5);
clilen=sizeof(cliaddr);
nsd=accept(sd,(struct sockaddr*)&cliaddr,&clilen);
if(nsd<0)
printf("Can't Accept\n");
else
printf("Accepted\n");
recv(nsd,recvmsg,100,0);
for(i=0;i<5;i++)
{
if(strcmp(recvmsg,oid[i])==0)
{
send(nsd,mdate[i],100,0);
send(nsd,time[i],100,0);
break;
}
}
}
```

MANAGER

```
#include<stdio.h>
#include<sys/types.h>
#include<netinet/in.h>
main()
{
int csd,cport,len,i;
char sendmsg[20],rcvmsg[100],rmsg[100],oid[100];
struct sockaddr_in servaddr;
printf("Enter the port\n");
scanf("%d",&cport);
csd=socket(AF_INET,SOCK_STREAM,0);
if(csd<0)
printf("Can't Create\n");
else
printf("Socket is Created\n");
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(cport);
if(connect(csd,(struct sockaddr*)&servaddr,sizeof(servaddr))<0)
                              printf("Can't Connect\n");
else
printf("Connected\n");
printf("\n 1.TCP Connection\n"); printf("\n 2. System \n");
printf("Enter the number for the type of information needed ... \n");
scanf("%d",&i);
if(i==1)
{
printf("Enter the Object ID for Client\n");
scanf("%s",oid);
send(csd,oid,100,0);
recv(csd,rmsg,100,0);
printf("\n The window size of %s is %s",oid,rmsg);
}
else
{
printf("\nEnter the Object ID for the System\n");
scanf("%s",oid);
send(csd,oid,100,0);
recv(csd,rmsg,100,0);
printf("\nThe Manufacturing date for %s is %s",oid,rmsg); recv(csd,rmsg,100,0);
printf("\nThe time of Last Utilization for %s is %s",oid,rmsg);
}
}
```

OUTPUT:

AGENT 1

```
[s@localhost ~]$ cc nex9ba1.c
[s@localhost ~]$ ./a.out
I'm the Agent - TCP Connection
Enter the Server port
_____________________
8100
Socket is Created
 Binded
Accepted
```

MANAGER

```
[s@localhost ~]$ cc nex9bman.c
[s@localhost ~]$ ./a.out
Enter the port
8100
Socket is Created
Connected
```

```
 1. TCP Connection
 2. System
Enter the number for the type of information needed....
1
Enter the Object ID for Client
client6
 The window size of client1 is 6
```

AGENT 2

```
[s@localhost ~]$ cc nex9ba2.c
[s@localhost ~]$ ./a.out
Enter the Server port
______________________
8200
Socket is Created

 Binded
Accepted
```

MANAGER

```
[s@localhost ~]$ cc nex9bman.c
[s@localhost ~]$ ./a.out
Enter the port
8200
Socket is Created
Connected
 1. TCP Connection
 2. System
Enter the number for the type of information needed....
2
Enter the Object ID for the System
System2
The Manufacturing date for System2 is 17.12.10
The time of Last Utilization for System2 is 11.30am
```

RESULT:

EX NO 13 **IMPLEMENTATIONOF SUBNETTING**

AIM: Write a java program to implement sub netting and find the subnet masks.

ALGORITHM:

1. Start the program
2. Declare the arguments as Host ip and Net mask
3. Calculate subnet-mask
4. Print the first address and last address
5. Compile and Run the program
6. Stop the program.

PROGRAM:

```
package subnet;
import java.util.Scanner;
import java.io.*;
import java.net.*;
public class Subnet {
public static void main(String[] args)
{
Scanner sc = new Scanner(System.in);
System.out.println("Enter the ip address");
String ip = sc.nextLine();
String split_ip[]=ip.split("\\.");

//SPlit the string after every .
String split_bip[] = new String[4];

//split binary ip
String  bip="";
for(int i=0;i<4;i++){
split_bip[i] = appendZeros(Integer.toBinaryString(Integer.parseInt(split_ip[i])));

// “18” => 18 => 10010=>00010010
bip += split_bip[i];
}
System.out.println("IP in binary is" +bip);
System.out.println("Enter the number of addresses:");
int n = sc.nextInt();

//Calculation of mask
int bits = (int)Math.ceil(Math.log(n)/Math.log(2));

/*eg if address = 120, log 120/log 2 gives log to the base 2 => 6.9068, ceil gives us upper
integer */
System.out.println("Number of bits required for address=" +bits);
int mask = 32-bits;
System.out.println("The subnet mask is="+mask);

//Calculation of first address and last address
int fbip[]= new int[32];
for(int i=0; i<32;i++)
fbip[i] = (int)bip.charAt(i)-48;

//convert character 0,1 to integer 0,1
for(int i=31;i>31-bits;i--)

//Get first address by ANDing last n bits with 0
fbip[i] &= 0;
String fip[]={"","","",""};
for(int i=0;i<32;i++)
fip[i/8] = new String (fip[i/8] + fbip[i]);
System.out.print("First address is = ");
for(int i=0;i<4;i++)
{
System.out.print(Integer.parseInt(fip[i],2));
if(i!=3)
System.out.print(".");
}
System.out.println();
int lbip[] = new int[32];
for(int i=0; i<32;i++)
lbip[i] = (int)bip.charAt(i)-48;

//convert cahracter 0,1 to integer 0,1
for(int i=31;i>31-bits;i--)

//Get last address by ORing last n bits with 1
lbip[i] |= 1;
String lip[] ={"","","",""};
for(int i=0;i<32;i++)
lip[i/8] = new String(lip[i/8]+lbip[i]);
System.out.print("Last address is =");
for(int i=0;i<4;i++){
System.out.print(Integer.parseInt(lip[i],2));
if(i!=3)
System.out.print(".");
}
System.out.println();
}
static String appendZeros(String s){
String temp= "00000000";
return temp.substring(s.length())+ s;
}
}
```

OUTPUT:

```
Enter the ip address
128.56.25.6
IP in binary is100000000011100000001100100000110
Enter the number of addresses:
5
Number of bits required for address=3
The subnet mask is=29
First address is = 128.56.25.0
Last address is =128.56.25.7
```

RESULT: